UNIVERZITA PARDUBICE

Fakulta elektrotechniky a informatiky

VYUŽITÍ NEURONOVÉ SÍTĚ PRO OCR

Bc. Tomáš Pilný

Diplomová práce

2023

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2022/2023

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Tomáš Pilný**
Osobní číslo:	**I21308**
Studijní program:	**N0714A150005 Automatické řízení**
Téma práce:	**Využití neuronové sítě pro OCR**
Zadávající katedra:	**Katedra řízení procesů**

Zásady pro vypracování

Cílem diplomové práce je tvorba webové aplikace, která bude konvertovat ručně psaný text za pomocí neuronové sítě do editovatelné verze.
V teoretické části student provede rešerši aplikací se zaměřením na strojový převod ručně psaného textu do digitální podoby.
V praktické části student navrhne implementaci několika variant neuronových sítí pro převod ručně psaného textu do digitální podoby. Student v rámci praktické části provede komparaci testovaných variant se zaměřením zejména na přesnost odhadu. Implementovaná neuronová síť bude schopna odlišit ručně psaný text od obrázků v rámci stanovených pravidel.

Rozsah pracovní zprávy: **60**
Rozsah grafických prací:
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

VONDRÁK, Ivo. Umělá inteligence a neuronové sítě. 3. vyd. Ostrava: VŠB – Technická univerzita Ostrava, 2009. ISBN 978-80-248-1981-5.
BERÁNEK, Ladislav. Neuronové sítě. České Budějovice: Jihočeská univerzita, 2007. ISBN 978-80-7394--050-8.
DOLEŽEL, Petr. Úvod do umělých neuronových sítí [online]. Pardubice: Univerzita Pardubice, 2016 [cit. 2022-10-17]. ISBN 978-80-7560-022-6. Dostupné z: https://e-shop.upce.cz/epub/

Vedoucí diplomové práce: **Ing. Daniel Honc, Ph.D.**
Katedra řízení procesů

Datum zadání diplomové práce: **8. listopadu 2022**
Termín odevzdání diplomové práce: **19. května 2023**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

Ing. Daniel Honc, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 15. listopadu 2022

Prohlášení autora

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 14. 08. 2023

Bc. Tomáš Pilný

Poděkování

Rád bych tímto poděkoval Ing. Danielu Honcovi, Ph.D. za vedení diplomové práce. Dále bych rád poděkoval Ing. Miroslavu Dvořákovi, Ing. Ladislavu Beranovi Ph.D. a doc. Ing. Petru Doleželovi, Ph.D. za cenné teoretické i praktické rady.

Rovněž bych rád poděkoval své sestře Mgr. Haně Pilné za pomoc při jazykové úpravě práce. V neposlední řadě bych rád poděkoval přátelům za mentální podporu po dobu tvorby práce.

V Pardubicích dne 14. 08. 2023

Bc. Tomáš Pilný

ANOTACE

Cílem této diplomové práce je vytvořit aplikaci, která pomocí webového rozhraní umožní uživateli nahrát obrázek ručně psaného anglického textu, který se následně odešle na server, kde dojde k jeho transformaci do digitální editovatelné verze pomocí neuronové sítě. Teoretická část je věnována rešerším aplikací a metodik transformace psaného textu do digitální podoby.

KLÍČOVÁ SLOVA

Neuronová síť, Umělá neuronová síť, CNN, OCR, Python, Tensorflow, WSL, React

TITLE

USAGE OF ARTIFICIAL NEURAL NETWORK FOR OCR

ANNOTATION

The objective of this thesis is to create an application that will allow users to upload images of handwritten text via a web interface. The image is then to be processed on the server and transformed into an editable digital form with use of a neural network. The theoretical part of the thesis is dedicated to the research of applications and methodology of transforming handwritten text into an editable digital version.

KEYWORDS

Neural network, Artificial neural network, CNN, OCR, Python, Tensorflow, WSL, React

OBSAH

SEZNAM ZKRATEK

2D	Two Dimensions	Dvourozměrný prostor
3D	Three Dimensions	Trojrozměrný prostor
AdaGrad	Adaptive Gradient Algorithm	Algoritmus adaptivního gradientu
Adam	Adaptive Moment Estimation	Adaptivní odhad momentu
AdamW	Adam with Weight Decay	Algoritmus Adam s úpadkem váhy
AI	Artificial Intelligence	Umělá Inteligence
ASGI	Asynchronous Server Gateway Interface	Brána rozhraní asynchronního webového serveru
BERT	Bidirectional Encoder Representations from Transformers	Obousměrné kódovací reprezentace z transformátorů
CNN	Convolutional Neural Network	Konvoluční neuronová síť
CPU	Central Processing Unit	Procesor
CUDA	Compute Unified Device Architecture	Typ softwarové a hardwarové architektury pro GPU
DOM	Document Object Model	Objektový model dokumentu
DVS	-	Dopředná vícevrstvá síť
EEG	Electroencephalography	Elektroencefalografie
EKG	Electrocardiography	Elektrokardiografie
GPT4	Generative Pretrained Transformer 4	Generativní předtrénovaný transformer 4
GPU	Graphical Processing Unit	Grafická karta
GRU	Gated Recurrent Unit	Branou řízená rekurentní jednotka
HTML	Hypertext Markup Language	Značkovací jazyk pro tvorbu webových stránek
ICR	Intelligent Character Recognition	Inteligentní rozpoznávání znaků
IWR	Intelligent Word Recognition	Inteligentní rozpoznávání slov
JPG/JPEG	Joint Photographic Experts Group	Formát obrazu se ztrát. kompresí
JSON	JavaScript Object Notation	Objektová JavaScript Notace
JSX/TSX	JavaScript XML / TypeScript XML	JavaScript XML / TypeScript XML
LLM	Large Language Model	Velký Jazykový Model
LSTM	Long Short Term Memory	Dlouhá krátkodobá paměť

NAdam	Nesterov-accelerated Adam	Nesterovem zrychlený Adam algoritmus
NPM	Node Package Manager	Správce balíčků pro JavaScript
OCR	Optical Character Recognition	Optické rozpoznávání znaků
OMR	Optical Mark Recognition	Optické rozpoznávání značek
OOP	Object Oriented Programming	Objektově orientované programování
OWR	Optical Word Recognition	Optické rozpoznávání slov
PDF	Portable Document Format	Přenosný dokumentový formát
PNG	Portable Network Graphics	Přenosná síťová grafika
REST API	Representational State Transfer Application Programming Interface	Reprezentativní přenos stavu aplikačním programovacím rozhraním
RGB	Color model composed from Red, Green and Blue color	Barevný model skládající se z červené, zelené a modré barvy
RMSprop	Root Mean Square Propagation	Kořenová střední kvadratická propagace
RNN	Recurent neural network	Rekurentní neuronová síť
ROI	Region Of Interest	Oblast zájmu
SCSS	Syntactically Awesome Style Sheets	Preprocesorový skriptovací jazyk kompilovaný do kaskádových stylů
SGD	Stochaistic Gradient Descent	Stochastický gradientní sestup
SPA	Single Page Application	Jednostránková webová aplikace
TXT	Plain textfile format	Textový soubor
UI	User Interface	Uživatelské rozhraní
URL	Uniform Resource Locator	Webová adresa
WSGI	Web Server Gateway Interface	Brána rozhraní webového serveru
WSL2	Windows Subsystem For Linux 2	Linuxový subsystém pro Windows 2
XML	Extensible Markup Language	Obecný značkovací jazyk

SEZNAM ILUSTRACÍ

ÚVOD

Optické rozpoznávání znaků neboli OCR (z anglického „Optical Character Recognition) je technologie, která hraje klíčovou roli v digitálním světě. Schopnost strojů převést ručně psaný text do digitální podoby má celou řadu využití. Mezi nejvýznamnější se řadí digitalizace ručně psaných dokumentů jako digitální archivování, automatizované zpracování formulářů, ale také například pomoc při vývoji autonomních vozidel a jejich schopnosti registrovat a rozeznávat dopravní značení a nápisy.

Současné výzkumy se soustředí na zlepšení přesnosti a efektivity OCR technologií, včetně využití pokročilých technik strojového učení a neuronových sítí. Přestože byly dosaženy značné pokroky, konverze ručně psaného textu zůstává zásadní výzvou, zejména vzhledem k rozdílům stylu písma a psaní jednotlivých osob. Nejjednodušší forma OCR je tak převod tištěného počítačového písma, díky jeho konzistentnímu a jednotnému stylu.

Vývoj webové aplikace, která může efektivně konvertovat ručně psaný text do editovatelné digitální podoby, tak může být cenným příspěvkem k řešení reálných problémů v různých oblastech od vzdělávání a vědy po obchod a průmysl. Tato práce se snaží přispět k tomuto dynamicky se rozvíjejícímu poli, navrhuje a testuje několik variant neuronových sítí pro účely OCR ručně psaného (tiskacího) písma a poskytuje tak nové poznatky a nástroje pro další výzkum a vývoj v této oblasti. Aplikace je omezena na detekci anglického textu z důvodu nenalezení potřebného datasetu ručně psaných českých písmen. Na tuto práci je tedy nutno nahlížet s přihlédnutím k faktu, že se rozsahem jedná o kombinaci tří témat, kde každé z nich by pokrylo samostatnou závěrečnou práci. Těmito tématy jsou:

- Problematika zpracování a segmentace obrazu;
- Problematika neuronových sítí a jejich aplikace na rozpoznání ručně psaných znaků;
- Kombinace obou předchozích problematik a jejich transformace do funkční webové aplikace.

1 OPTICKÉ ROZPOZNÁVÁNÍ ZNAKŮ

Optické rozpoznávání znaků, o kterém se někdy mluví primárně jako o optickém rozpoznání textu je metoda separace textu z psaného nebo tištěného dokumentu (například pomocí scanneru nebo ve formě obyčejné fotografie) a jeho následné převedení do elektronické podoby pomocí specializovaného softwaru (IBM Cloud education, 2022).

Optické rozpoznávání znaků se dělí do tří skupin: OCR, ICR, OMR. Základní forma je OCR, která využívá databázi různých elektronických fontů a obrázků textu, nad nimiž provádí tzv. „pattern-matching. Naskenovaná fotka se převede na bitmapu tvořenou hodnotami 0 a 1, kde 0 odpovídá bílé barvě pozadí a 1 odpovídá černé barvě textu. Tato bitmapa se potom porovnává s databází a na základě shody rozhodne o jaké písmeno nebo číslici se jedná. V případě, že nedochází ke skenování jednotlivých písmen, ale celých slov, se hovoří o tzv. „Optical Word Recognition“ (OWR). Hlavní nevýhodou těchto dvou metod je, že databáze musí být velmi rozsáhlá, protože existuje nespočet různých fontů (IBM Cloud education, 2022).

Obr. 1 - Převedení písmena na bitmapu (zjednodušený příklad) (zdroj: vlastní)

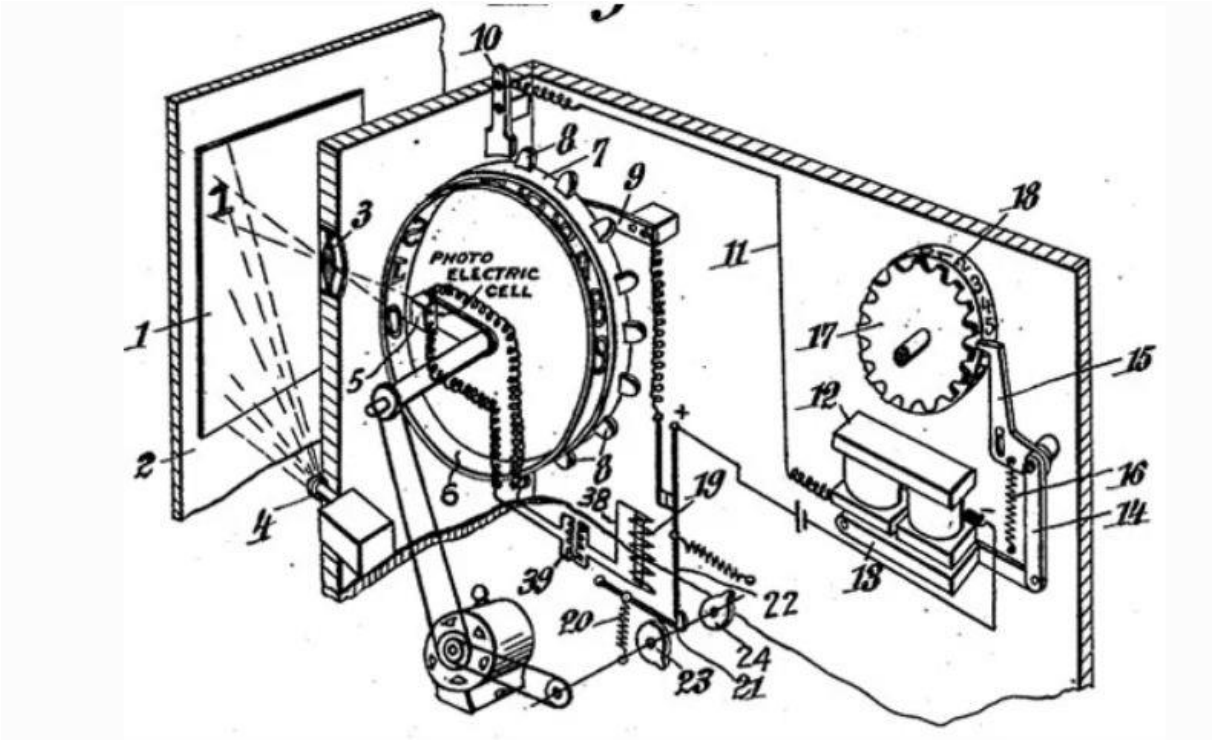
Druhou variantou je tzv. Inteligentní rozpoznávání znaků neboli ICR („Intelligent Character Recognition“). Je to způsob, který je využíván moderními systémy, a který implementuje strojové učení a neuronové sítě. Neuronová síť, která je korektně vytrénovaná na rozeznání jednotlivých znaků na předem připraveném datasetu umožňuje rozeznat i hůře čitelné znaky psané ručně. Také tato varianta existuje ve verzích, kdy pracuje jak s jednotlivými znaky, tak s celými slovy. V takovém případě se jedná o „Intelligent Word Recognition“ neboli IWR (AWS, 2023).

Poslední variantou je tzv. optické rozpoznávání značek neboli OMR („Optical Mark Recognition“), což je způsob identifikace značek (log), průsvitek a dalších netextových symbolů v dokumentu (AWS, 2023).

Tato práce, ač nese název „Využití neuronové sítě pro OCR“, se zabývá právě ICR (OCR je často zaměňováno za ICR), konkrétně jeho formou rozpoznávající jednotlivé znaky a nikoliv slova. Z teoretického hlediska by bylo zajímavé srovnání kvality překladu mezi ICR a IWR variantami.

1.1 HISTORIE A VÝVOJ OCR

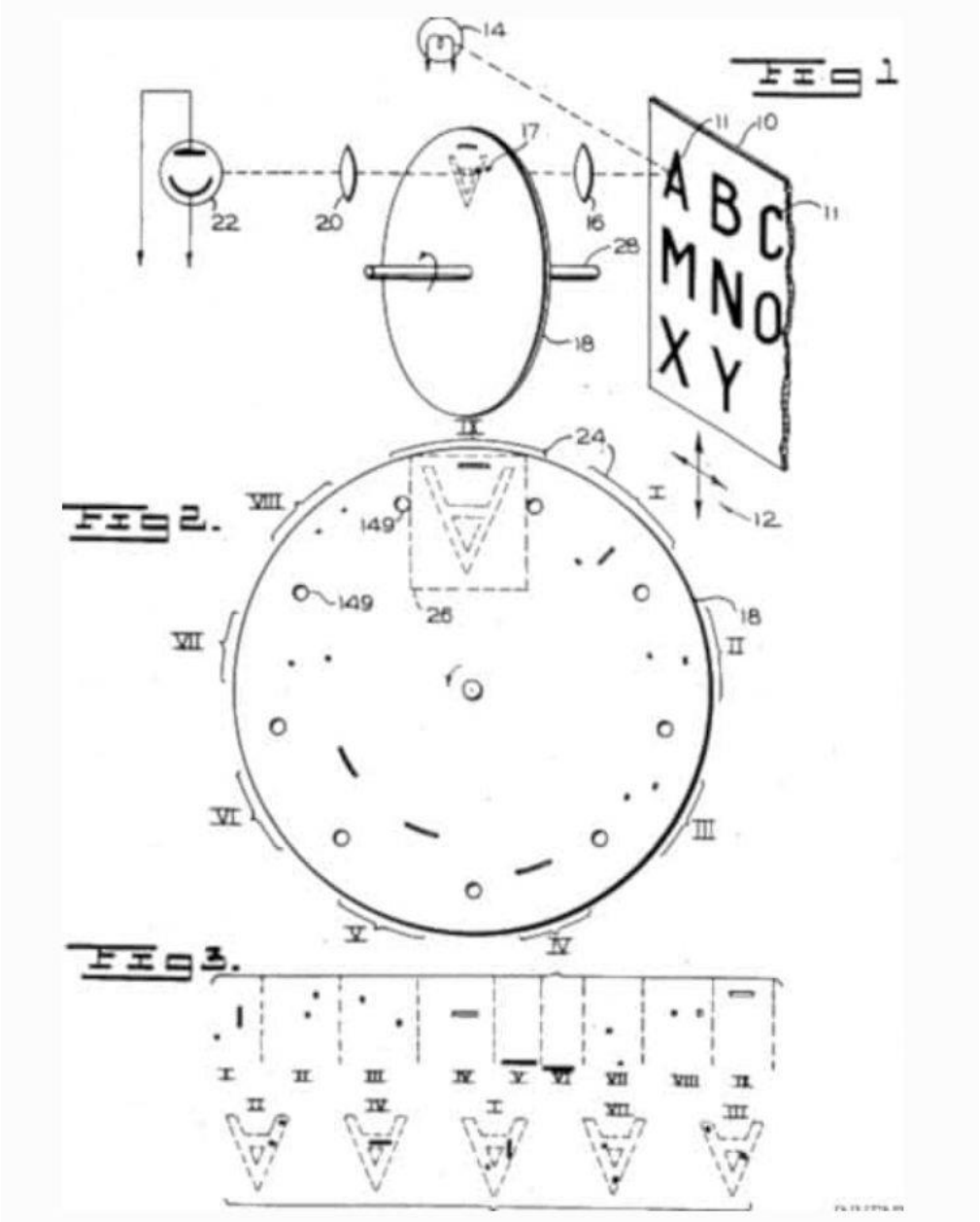
Počátky vývoje OCR sahají do roku 1935 k Rakouskému vynálezci Gustavu Tauschekovi. Tomu byl v Americe udělen patent za jeho vynález s názvem „Reading Machine of Tauschek“ neboli „Tauschekův čtecí stroj“. Jednalo se o mechanické zařízení, které bylo schopno z obrazu přečíst písmena a číslice a vytisknout je na papír. Zařízení používalo šablonu, která se shodovala s fotoelektrickým fotodetektorem. Když obraz procházel před okénkem, které sloužilo jako „oko“, tak se porovnávací zařízení ve formě disku s dírami ve tvaru čísel a písmen otáčelo před okénkem a hledalo shodu. Pokud se text v obrázku shodoval s jednou z děr na porovnávacím zařízení, pak stroj pomocí tiskového bubnu vytiskl dané písmeno (Heather, 2022).



Obr. 2 - Kresba Tauschekova čtecího stroje (History-computer.com, 2022)

V roce 1951 Americký vynálezce a kryptoanalytik David Hammond Shepard představil první komerčně dostupný OCR systém s názvem GISMO. Toto zařízení bylo schopno rozeznat 23 písmen latinské abecedy – což odpovídalo znakové sadě psacího stroje v té době. Po dalším roce práce a investic byl přístroj schopen rozeznat již všech 26 písmen abecedy a v roce 1953 získal Shepard na svůj přístroj patent, o který zažádal v roce 1951. Shepard taktéž vynalezl font, který byl jednodušší na detekování jeho přístrojem. Prvotně jeho přístroj fungoval tak, že pomocí fotoelektrického senzoru skenoval jednotlivá latinská písmena. Následně přístroj nahrál obsah toho, co přečetl na děrný štítek, a to na základě podobnosti znaků, uložených v jeho

vestavěné paměti. V pozdější fázi fotoelektrický senzor nahradila laserová technologie (History Computer Staff, 2023).



Obr. 3 - Patentová kresba principu přístroje GISMO (History Computer Staff, 2023)

V roce 1974 představil Americký vynálezce Ray Kurzweil svůj přístroj, který byl schopný převést text v prakticky jakémkoliv soudobém standartním fontu do elektronické podoby. To představovalo velký posun oproti předešlé technologii, která vyžadovala specifický font. Jeho cílem bylo umožnit převodem textu do mluveného slova, čtení dokumentů pro nevidomé lidi. V roce 1980 Kurzweil prodal svoji firmu firmě Xerox, díky které stoupla

popularita OCR okolo roku 1990 digitalizací historických novinových článků (IBM Cloud Education, 2022).

1.2 SOUČASNÉ PŘÍSTUPY OPTICKÉHO ROZPOZNÁNÍ ZNAKŮ

Existuje několik variant klasifikačních metod ručně psaného písma. Klasifikací se rozumí proces, kdy se daný matematický model / systém „naučí“ tvary vstupních dat a na základě konkrétní metody určí, o jaký znak se jedná. V následující části bude rozebráno několik nejpoužívanějších variant optického rozpoznávání znaků současného století.

1.2.1 Technika porovnávání šablon („Template matching“)

Jedná se pravděpodobně o nejjednodušší formu optického rozpoznávání znaků. Tato technika, jak název napovídá, využívá porovnání mezí vstupním obrazem (typicky bitmapou) a databází vzorů, ve které je uloženo množství písmen a číslic různých fontů. Tento přístup je podobný původním způsobům OCR, zmíněným v kapitole 1.1 . Princip je následující: algoritmus posouvá šablonu znaku přes cílový obraz (jedná se o algoritmus posuvného okna, který je blíže popsán v kapitole 1.4.3) a v každém bodě porovnává šablonu s částí vstupního obrazu. Při každém posunutí se vypočte míra shody mezi šablonou a odpovídající části vstupního obrazu. Možnými metodami pro výpočet míry shody je například křížová korelace, metoda nejmenších čtverců nebo Euklidovská vzdálenost. Pokud je míra shoda vyšší než hraniční hodnota, pak se identifikuje shoda mezi znakem a vzorem (Memon, 2020).

Nevýhodou této techniky je, že ač je schopna poměrně spolehlivě rozeznávat počítačové (tiskové) písmo, což odpovídá klasickému OCR, tak není efektivně použitelná pro ICR a rozeznání ručně psaného písma. Může za to fakt, že každé ručně psané písmo je trochu jiné. Databáze uložených vzorů by tak musela být značně rozsáhlá, což vede k problémům s paměťovou a výpočetní náročnosti. Pro zlepšení detekce se v takovém případě používá technika, která zdeformuje písmo a porovná jej specificky s databází zdeformovaných znaků (Memon, 2020).

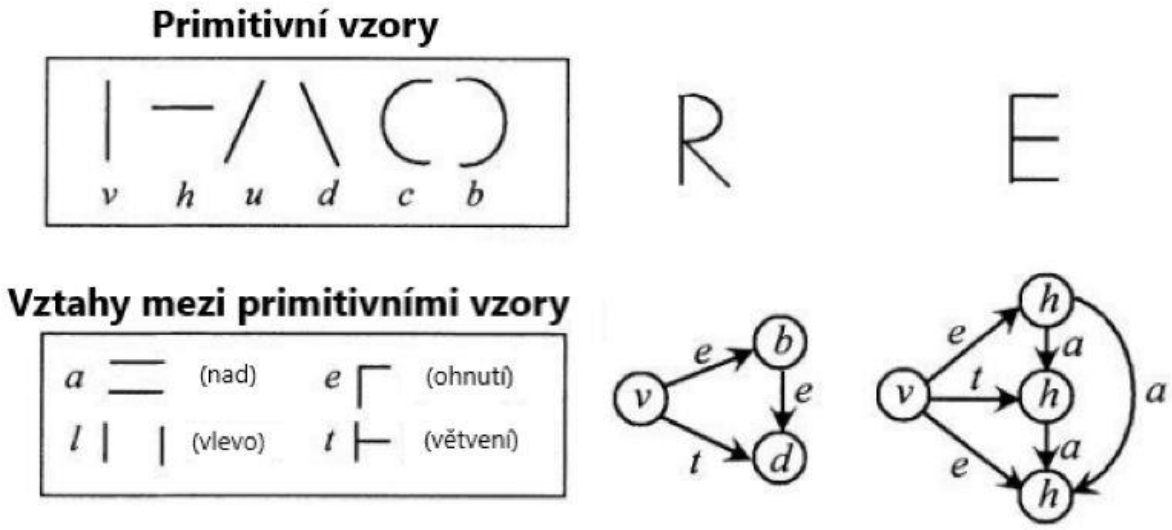
1.2.2 Technika strukturálního porovnání („Structural Pattern Recognition“)

Tato technika se zaměřuje na vztahy mezi jednotlivými částmi vzoru. Zatímco technika porovnání šablon a statistické metody se zaměřují na porovnání celkového vstupního obrazu se vzorem, tato metoda porovnává jednotlivé části vstupního obrazu a jakým způsobem jsou spojeny. Je tedy nutné vstupní obraz rozložit na skupinu primitivních vzorů. Toho je možné docílit například použitím histogramu řetězcového kódu („chain code histogram“), což je

způsob popisu kontury neboli obrysu objektu pomocí sekvence symbolů nebo čísel. Histogram zobrazuje frekvence jednotlivých směrů v řetězcovém kódu. Představuje tak distribuci směrů v rámci kontury objektu, což umožňuje robustní reprezentaci tvaru. Aby bylo možné tuto metodu aplikovat, musí být obraz převeden na bitmapu. Tato metoda má taktéž své limitace pro rozpoznávání ručně psaného písma. Stačí aby písmena byla propojena nebo aby obraz neměl v binární podobě dostatečnou kvalitu. Existují dvě varianty této metody, a sice: metoda grafů a metoda založená na gramatice (Memon, 2020).

Grafy se v úlohách optického rozpoznávání znaků využívají k matematickému popisu objektů. Využívají k tomu uspořádané dvojice uzlů (N) a hran (E). Hrany v tomto případě většinou reprezentují oblouk psacího tahu písma, propojující jednotlivé uzly. Konkrétní uspořádání pak definuje, o jaké konkrétně písmo nebo číslici se jedná. Grafy mohou být neorientované nebo orientované. K vyhodnocení podobnosti grafů mezi vstupním obrazem a vzorem lze využít například algoritmu metody podobnosti vrcholů nebo SimRank algoritmu (Memon, 2020).

Metoda založená na gramatice rovněž využívá matematického popisu pomocí grafů. Zde se grafy využívají k syntaktické analýze, která zjišťuje podobnosti ve strukturách primitivních vzorů (Memon, 2020).



Obr. 4 – Popis písmen pomocí řetězcového kódu (převzato a přeloženo z (Memon, 2020))

1.2.3 Statistické metody porovnání

Tyto metody využívají matematické a statistické techniky k analýze a interpretaci dat obsažených v obraze. Je možné je dělit na parametrické a bezparametrické. Parametrické metody předpokládají, že data pocházejí z určitého známého pravděpodobnostního rozdělení. Jsou založeny na parametrech, které charakterizují rozdělení, jako je průměr a směrodatná odchylka v případě normálního rozdělení. Spadají sem algoritmy jako například: logistická regrese, lineární diskriminační analýza a další. Parametrické metody je obecně vhodné použít v případě, kdy víme, že data odpovídají určitému rozdělení. Tyto informace pak lze využít k efektivnější analýze (Memon, 2020).

Bezparametrické metody nepředpokládají, že by data pocházela ze známého pravděpodobnostního rozdělení. To jim poskytuje vyšší flexibilitu a umožňuje jejich aplikaci na širší škálu dat, ale jejich složitost s množstvím vstupních dat narůstá. Do této kategorie spadají například rozhodovací stromy nebo algoritmus Kého nejbližšího souseda (Memon, 2020).

1.2.4 Metody porovnání využívající jádro („Kernel methods“)

Tyto metody se skládají ze dvou částí. První část provádí mapování dat do vyššího dimenzionálního prostoru, který se někdy nazývá Vlastnostní prostor. Tato transformace umožňuje modelu pracovat s daty, která mohou být v původním prostoru nelineární, jako by byla lineární. Druhou částí je učící algoritmus, který slouží k detekci lineárních vzorů ve Vlastnostním prostoru. Tyto metody využívají tzv. jádrovou funkci („kernel function“), která je zodpovědná za provádění výpočtu v tomto vyšším dimenzionálním prostoru, aniž by bylo nutné data v tomto prostoru explicitně zobrazovat. Pro lepší představení, jak tyto metody fungují, si lze představit, že jsou vstupní data „zamotaná“ a nečitelná v běžném dvourozměrném (2D) prostoru. Aby bylo možné tato data efektivně interpretovat a detekovat v nich vzory, tak se musí transformovat do vícerozměrného prostoru. Výpočty nad takovými daty by však byly značně výpočetně náročné, a proto se využívají jádrové funkce. Tento princip lze aplikovat právě například na ručně psané písmo, které může mít mnoho složitých tvarů a křivek. Tohoto principu se využívá například u Podpůrných vektorových strojů („Support Vector Machines“) (Shawe-Taylor, Christianini, 2004).

1.2.5 Metody využívající umělé neuronové sítě („artificial neural networks“)

Metody využívající umělých neuronových sítí patří mezi nejefektivnější způsoby optického rozpoznávání psaného písma. Jejich fungování a struktura je inspirována stavbou a funkcí lidského mozku. Tyto metody modelují složité vzory a interakce mezi daty prostřednictvím souboru propojených uzlů, které zpracovávají a předávají informace. Tyto uzly jsou uspořádány do vrstev a pracují společně, aby odhadly komplexní funkce, rozpoznaly vzory nebo provedly komplexní klasifikační úlohy. Neuronové sítě v současnosti představují klíčovou složku v moderním strojovém učení a jsou základem mnoha aplikací. Kromě klasifikačních úloh se neuronové sítě využívají například i pro zpracování přirozeného jazyka nebo při vývoji autonomního řízení vozidel. Tato práce se podrobněji věnuje popisu neuronových sítí v kapitole 2 (Memon, 2020).

1.3 PŘEDZPRACOVÁNÍ OBRAZU

V následující části se práce věnuje již pouze metodám a postupům, které se uplatňují v rámci implementace ICR ručně psaného textu v praktické části.

Nejdříve je nutné převést požadovaný ručně psaný text do elektronické podoby. To je možno udělat buď s využitím skeneru nebo pořízením fotografie. Počítač se chová k obrazu, který je v tento moment uložen ve formátu JPG nebo PNG, jako k vícerozměrnému poli (matici), pokud se jedná o černobílý obraz (grayscale). Každá buňka matice se nazývá pixel a je v ní uložena 8bitová celočíselná hodnota v rozsahu 0–255. Barevný obraz, je pak reprezentován pomocí tří matic, kde jednotlivé matice odpovídají složkám RGB spektra. V případě barevného obrazu, má každý pixel tři 8bitové hodnoty (pro každý kanál jeden) a jejich kombinací vznikají další barvy. Výsledný obraz pro potřeby optického rozpoznání by měl být ve formě bitmapy (každý bod obrazu nabývá hodnoty 0 nebo 1). Tím se obraz rozdělí na popředí, kde se nachází písmo a pozadí (Reddy, 2019).

Předtím než je možné využít neuronovou síť pro rozeznání jednotlivých písmen, je nutné tato písmena z textu získat – jedná se o tzv. segmentaci obrazu čili rozdělení obrazu na části, které budou předloženy na vstup neuronové sítě. Před samotnou segmentací obrazu je potřeba obraz ručně psaného písma předzpracovat. Předzpracování obrazu hraje klíčovou roli v kvalitě a přesnosti výsledku a je stejně důležité jako samotná kvalita vytrénování a robustnosti modelu neuronové sítě, která se použije k odhadu jednotlivých znaků. Cílem předzpracování obrazu je tak odstranit šum, nežádoucí zkreslení, a naopak zvýraznit obrysy jednotlivých písmen, aby je bylo možné jednodušeji rozeznat. Samotné předzpracování obrazu se skládá z několika na sobě nezávislých kroků, které je ale potřeba provést v patřičném pořadí.

1.3.1 Stínování (grayscaling) obrazu

Grayscaling je metoda při níž se barevný obraz převede do šedé barvy, kde je každý pixel reprezentován 8bitovou celočíselnou hodnotou. Výsledkem je tak stále ostrý obraz v různých odstínech šedivé barvy kde hodnota 0 odpovídá černé barvě a hodnota 255 odpovídá bílé barvě. Jelikož cílem OCR je pracovat s písmem, není důležité, aby bylo barevné, a grayscaling současně umožní snížit velikost obrazu. Obrázek se převede tak, že se RGB hodnoty jeho jednotlivých pixelů zprůměrují. Nelze však použít obyčejný aritmetický průměr, protože lidské oko vnímá jednotlivé barevné složky s různou intenzitou. Je tedy potřeba použít vážený průměr, kterým se zajistí vyrovnání tohoto faktu.

Knihovna OpenCV, která bude zmíněna později a která byla použita v rámci vytváření aplikace, implementuje grayscaling následovně:

$$Grayscale = \frac{(0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B)}{3} \quad (1)$$

Kde:

- R, G, B označují celočíselné hodnoty jednotlivých barevných kanálů

(Hughes, 2022).

Výsledný obraz pak vypadá následovně:

Obr. 5 - Ukázka obrázku po jeho převedení do grayscale (zdroj: vlastní)

1.3.2 Binarizace obrazu

Jakmile je obrázek převeden do grayscale podoby, je potřeba provést jeho binarizaci, která se taky označuje jako prahování (z anglického „thresholding“). Jedná se o proces, kdy se obraz převede na bitmapu. Jak bylo zmíněno dříve, tato metoda se využívá k rozdělení obrazu na popředí a pozadí a umožňuje tak z obrazu vyseparovat důležité informace. Převod funguje na základě porovnání individuálních hodnot pixelů s prahovou hodnotou. Pokud je hodnota pixelu nižší než prahová hodnota, pixel se nastaví na hodnotu 0 v opačném případě na hodnotu 255. Existují dva druhy prahování: lokální a globální (Piyadasa, 2022).

V případě globální binarizace se použije jediná hodnota nastavená uživatelem pro každý pixel v obrazu. Tato metoda má nevýhodu v tom, že není schopna zohlednit různé nasvícení obrazu. Používanějším a efektivnějším způsobem pro binarizaci obrazu s nehomogenním nasvícením je Otsuova metoda pojmenována po jejím autorovi, japonském matematikovi Nobuyuki Otsuovi. Tato metoda využívá histogram obrazu k nalezení prahové hodnoty, která minimalizuje intra-třídní rozptyl („weighted within-class variance“). Pro vysvětlení uvažujme obraz s intenzitou pixelů v rozsahu od 0 do L–1 (pro 8bitový obraz L=256). Pokud je *t* definováno jako prahová hodnota, pak je možné rozdělit intenzitu pixelů na 2 třídy (popředí a pozadí). První třída (popředí) obsahuje pixely s intenzitou od 0 do *t*–1. Druhá třída (pozadí) obsahuje pixely s intenzitou od *t* do L–1 (Murzova, Seth, 2020).

Pro toto rozdělení je možné vypočítat pravděpodobnost, že náhodně vybraný pixel má intenzitu odpovídající hodnotě *i* následujícím způsobem.

$$p_i = \frac{n_i}{N} \tag{2}$$

Kde:

- p_i je pravděpodobnost, že pixel bude mít intenzitu *i;*
- n_i je počet pixelů s intenzitou *i;*
- N je celkový počet pixelů v obraze.

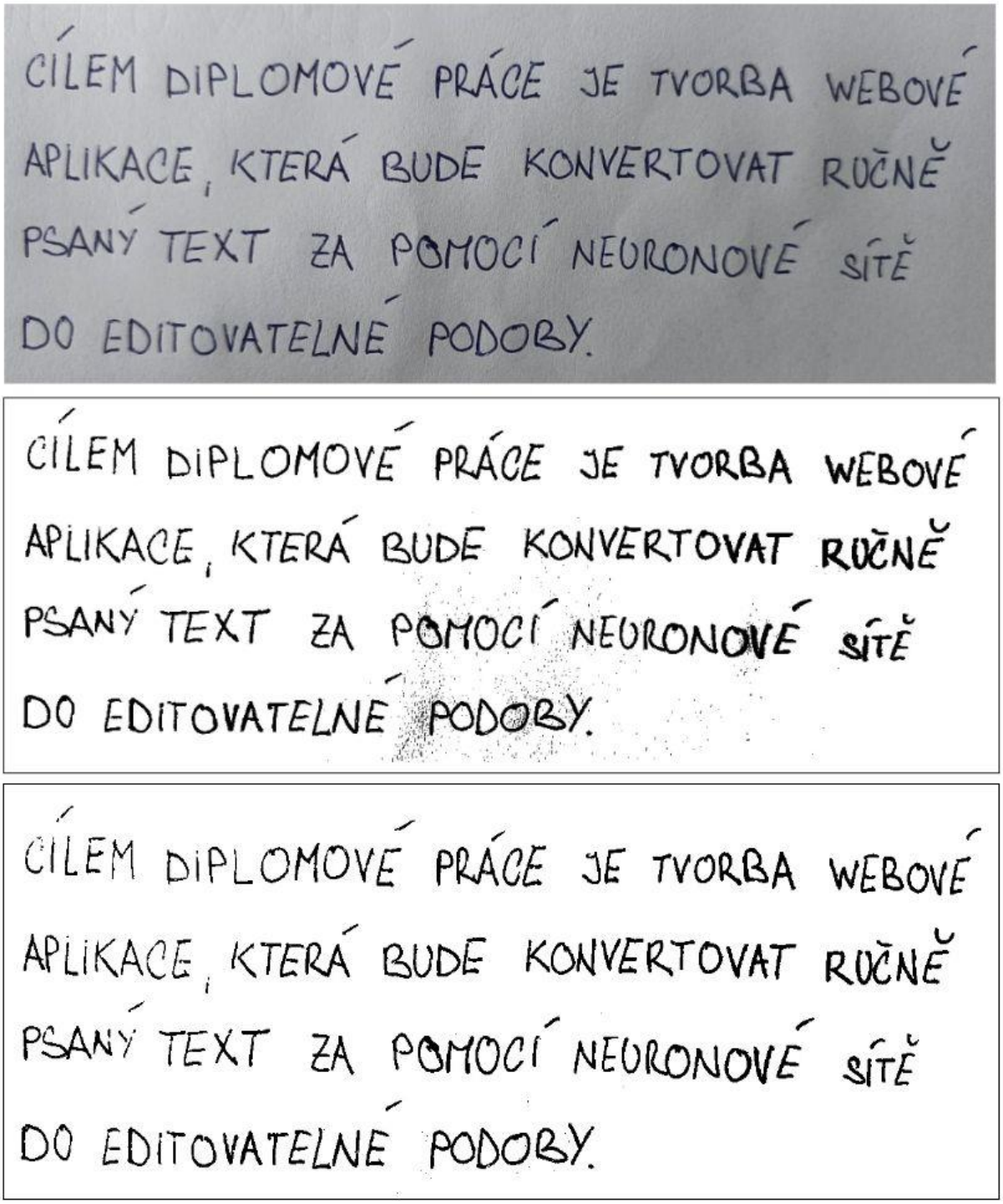
Cílem metody je tak nalézt hodnotu *t*, která minimalizuje σ^2. Toho je docíleno procházením všech možných hodnot *t* a výpočtem σ^2 pro každou z nich. Z těchto vypočtených hodnot se poté vybere *t*, které dává nejnižší hodnotu σ^2. Toto *t* je pak použito jako práh pro binarizaci obrazu. Matematicky lze výpočet intra třídního rozptylu, který je definován jako vážený součet rozptylů obou tříd, popsat následujícím vztahem:

$$\sigma^2(t) = w_1(t) \cdot {\sigma_1}^2(t) + w_2(t) \cdot {\sigma_2}^2(t) \quad (3)$$

Kde:

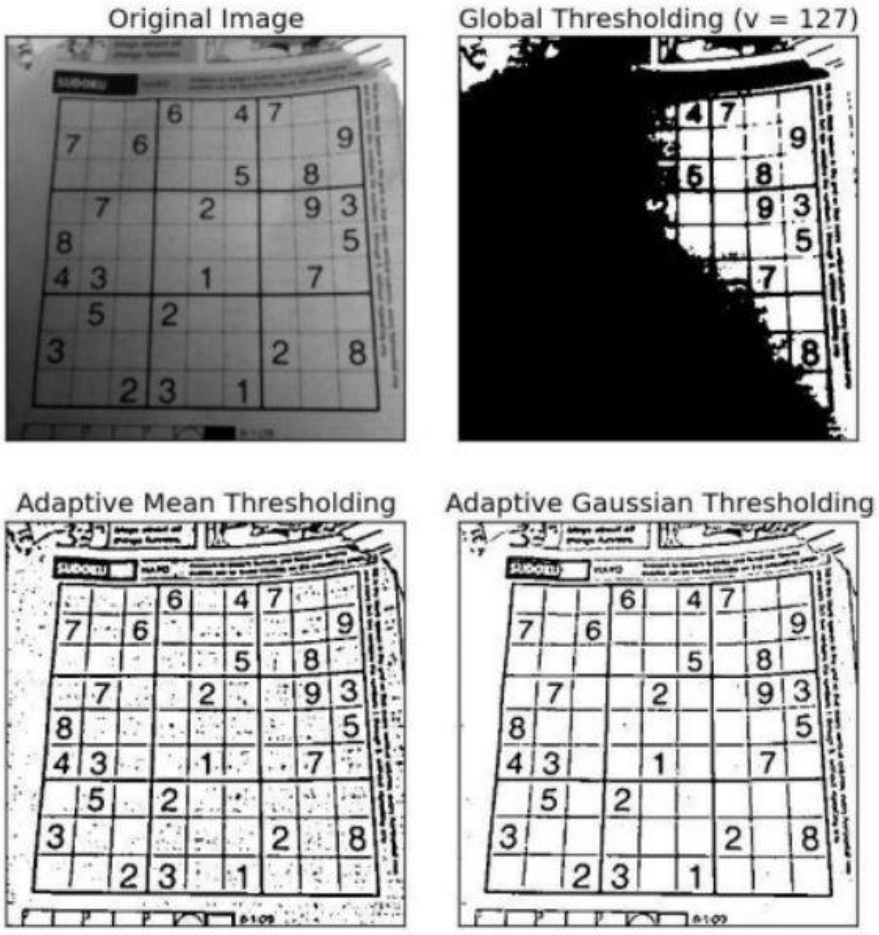
- $\sigma^2(t)$ je intra třídní rozptyl;
- $w_1(t)$ je celková pravděpodobnost, že náhodně vybraný pixel je součástí popředí a lze vyjádřit jako suma pravděpodobností p_i, pro všechny intenzity *i* od 0 do *t*-1;
- $w_2(t)$ je celková pravděpodobnost, že náhodně vybraný pixel je součástí pozadí a lze vyjádřit jako suma pravděpodobností p_i, pro všechny intenzity *i* od *t* do L-1;
- ${\sigma_1}^2(t)\ a\ {\sigma_2}^2(t)$ jsou rozptyly popředí a pozadí.

(Murzova, Seth, 2020).



Obr. 6 - Ukázka binarizace obrazu, sestupně: originální obraz, základní globální binarizace, Otsuova metoda (zdroj: vlastní)

Ač je Otsuova metoda kvalitnější než základní forma globální binarizace, tak pro případy náročnějších světelných podmínek (stíny, rozmazání části obrazu atd.) je stále nedostatečná. Proto se používají metody lokální binarizace. Jednou z nejpoužívanějších variant je adaptivní binarizace. Ta obraz rozdělí na oblasti o velikosti n x n pixelů a počítá hodnotu prahu pro každou oblast zvlášť. Jsou dva způsoby, kterými lze novou hodnotu vypočítat: dle průměru („Adaptive Mean Thresholding“) nebo podle Gaussovy funkce („Adaptive Gaussian Thresholding“). Adaptivní binarizace dle průměru vypočítá prahovou hodnotu jako průměr intenzity sousedních pixelů (v rámci oblasti n x n) a poté odečte předem stanovenou konstantu C (celočíselná hodnota). Tato metoda je vhodná, pokud jsou oblasti podléhající binarizaci relativně jednolité vzhledem k intenzitě. Adaptivní binarizace podle Gaussovy funkce vypočítá prahovou hodnotu jako vážený průměr intenzity sousedních pixelů, kde váhy jsou dané Gaussovou funkcí, a poté odečte konstantu C. Tato metoda je užívána pro obraz s různou úrovní detailu, protože dává větší váhu pixelům, které jsou blíže k pixelu, pro který se vypočítává prahová hodnota (Piyadasa, 2022).



Obr. 7 - porovnání různých binarizačních technik (OpenCV, 2023)

1.3.3 Rotace

Další důležitou součástí předzpracování obrazu je ošetření rotace nebo zkosení obrazu ("deskewing"). Zkosení a nežádoucí rotace může mít zásadní vliv na kvalitu OCR. Nejdříve je nutné detekovat úhel rotace, k čemuž lze využít několik metod. Příkladem je Houghova transformace, která detekuje přímky v obraze a na základě jejich orientace umožňuje detekovat úhel natočení. Jakmile je nalezen úhel rotace, je možné obraz pootočit o zápornou hodnotu tohoto úhlu. Lze též využít afinní transformaci. Jedná se o lineární transformaci, která se skládá ze dvou částí: rotace a translace. Rotační matice má v případě 3D prostoru tři různé varianty, pro každou osu jednu. V případě 2D prostoru, probíhá rotace okolo osy Z, viz rovnice (4) (Gonzales, Woods, 2008).

$$R(\theta) = \begin{bmatrix} \cos\,\theta & -\sin\,\theta \\ \sin\,\theta & \cos\,\theta \end{bmatrix} \tag{4}$$

Kde:

- $R(\theta)$ je rotační matice pro rotaci o úhel θ kolem počátku souřadnicového systému

$$T(tx, ty) = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

Kde:

- $T(tx, ty)$ je translační matice pro posun v ose X o tx a posun v ose Y o ty

Kombinace těchto dvou transformací, lze provést násobením těchto matic v homogenních souřadnicích. Nejdříve je tedy nutné rozšířit rotační matici do homogenních souřadnic:

$$R'(\theta) = \begin{bmatrix} \cos\,\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{6}$$

A následně je možné matice vynásobit:

$$T \cdot R' = \begin{bmatrix} \cos\,\theta & -\sin\theta & tx \\ \sin\theta & \cos\theta & ty \\ 0 & 0 & 1 \end{bmatrix} \tag{7}$$

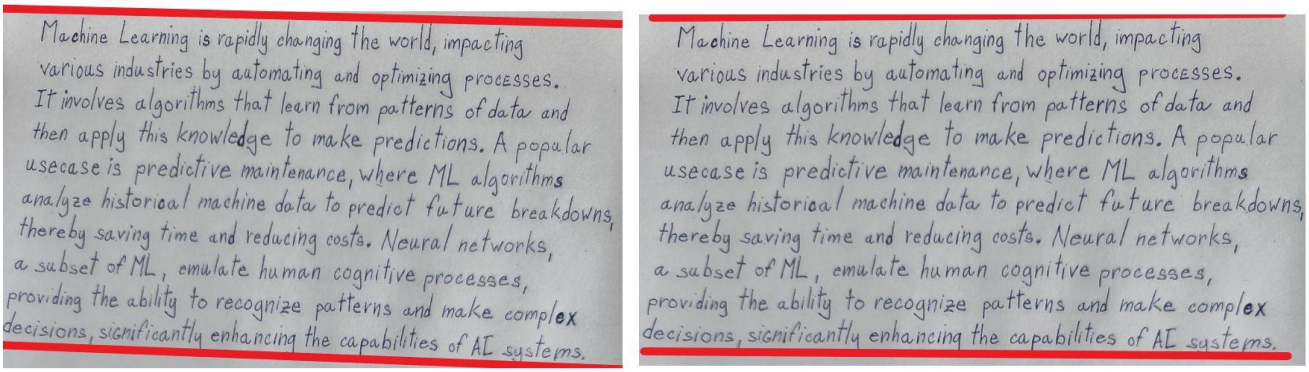
Tím, že se tato transformace aplikuje na souřadnice pixelů (x, y) v obraze, vzniknou nové souřadnice (x*, y*), reprezentující rotovanou pozici pixelu. Těchto hodnot je dosaženo následující úpravou:

$$\begin{bmatrix} x^* \\ y^* \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & tx \\ \sin\theta & \cos\theta & ty \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (8)$$

Kde:

- $\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ je vektor původních souřadnic pixelu;
- $\begin{bmatrix} x^* \\ y^* \\ 1 \end{bmatrix}$ je vektor nových souřadnic pixelu po rotaci.

Jelikož samotná transformace rotuje obraz kolem souřadnic (0, 0), tak je často nutné aplikovat translaci před a po rotaci, aby se obraz neposunul mimo pohled (Gonzales, Woods, 2008).



Obr. 8 - Ošetření nežádoucí rotace (vlevo původní obraz, vpravo obraz po korekci) (zdroj: vlastní)

1.3.4 Odstranění šumu

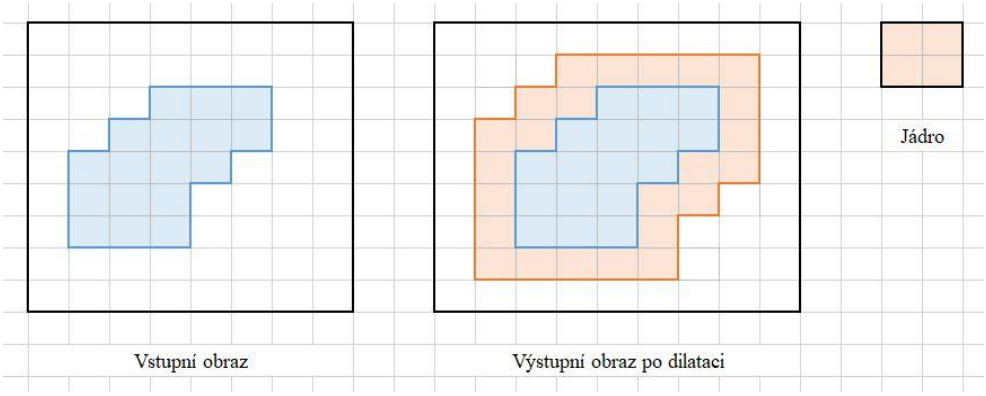
Odstranění šumu je kombinací několika technik, které mají za úkol z obrazu odstranit malé tečky (obecně se tomuto typu šumu říká „Salt & Pepper“), které by mohly kvůli vysoké intenzitě narušit kvalitu segmentace. Častou kombinací technik jsou například: dilatace, eroze, aplikace mediánových nebo Gaussovských filtrů a další morfologické operace. V biologii je termínem morfologie rozuměno struktura a složení tvorů a rostlin. Z matematického a informačně-technologického hlediska se jedná o operace využívané v počítačovém vidění k úpravě obrazu. Morfologické operace pracují s filtrem (jádrem), který aplikují na vstupní obraz s cílem vytvořit upravený obraz o stejných rozměrech. Filtr je čtvercová jednotková matice, která se postupně přikládá na každý pixel ve vstupním obraze. V rámci morfologických operací získává pixel ve výstupním obraze svoji hodnotu na základě porovnání pixelu ve vstupním obraze s jeho sousedními pixely (Chhikara, 2022).

Dilatace je operace, která má „rozšířit“ nebo „zvětšit“ objekty popsané bílými pixely v binárním obraze. Pokud filtr překrývá ve vstupním obraze bílý pixel, tak se ve výstupním obraze na bílou barvu nastaví všechny pixely, které se v daný moment nachází pod filtrem ve vstupním obraze. Dilataci je tak možné využít např, ke spojení mezer v rámci písmene. Matematicky lze aplikaci dilatace na vstupní obraz popsat následujícím vztahem (Gonzales, Woods, 2008).

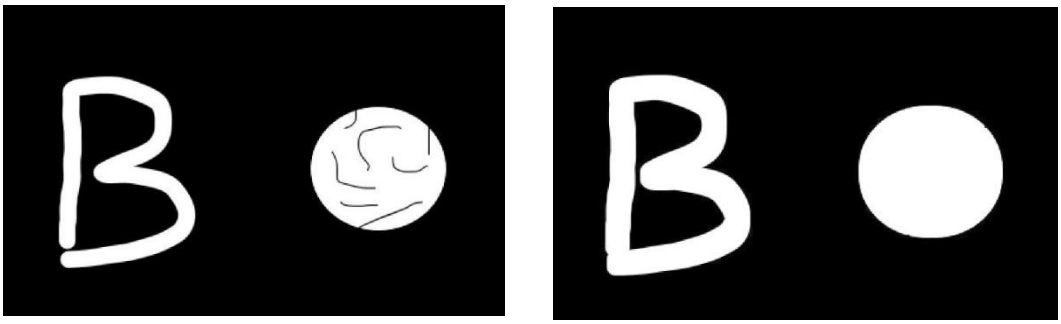
$$(A \oplus B)(x, y) = \max_{(s,t)\in B} \{A(x - s, y - t)\} \tag{9}$$

Kde:

- A je vstupní obraz;
- B je strukturální element (v tomto případě čtverec, ale může mít i jiné tvary);
- $\oplus$ označuje operaci dilatace;
- (s, t) jsou relativní pozice vzhledem k aktuálnímu bodu (x, y).



Obr. 9 - Princip dilatace obrazu (zdroj: vlastní)



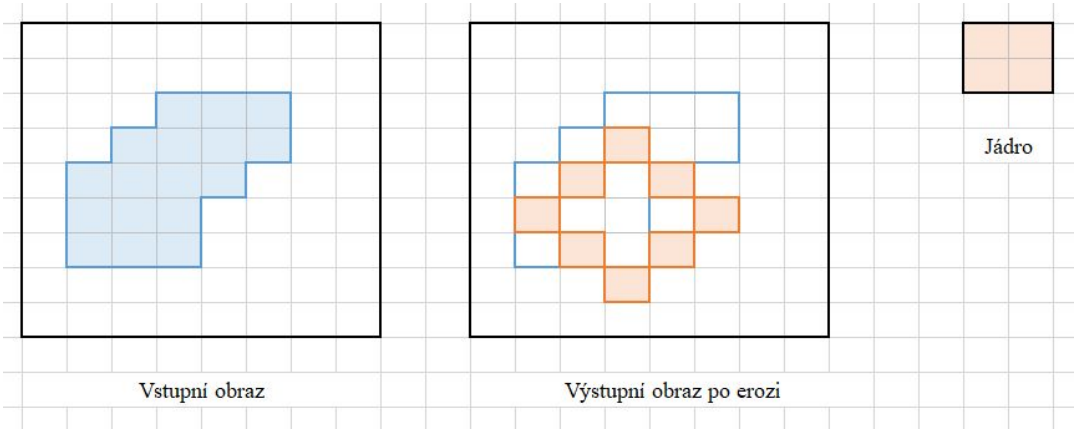
Obr. 10 – Obraz před dilatací (vlevo) a po dilataci (zdroj: vlastní)

Eroze má opačný význam a sice „zúžení“ nebo „zmenšení“ objektu. Pokud má daný pixel aspoň jeden sousední pixel černý (nastavený na hodnotu nula), tak se také nastaví na hodnotu nula. Matematicky lze aplikaci eroze na vstupní obraz popsat následujícím vztahem (Gonzales, Woods, 2008).

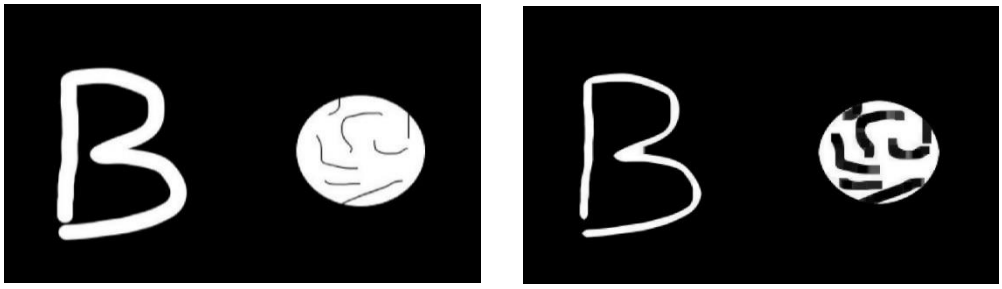
$$(A \ominus B)(x,y) = \min_{(s,t)\in B}\{A(x-s, y-t)\} \tag{10}$$

Kde:

- $\ominus$ označuje operaci eroze



Obr. 11 - Princip eroze obrazu (oranžové ohraničení označuje výsledný obraz) (zdroj: vlastní)



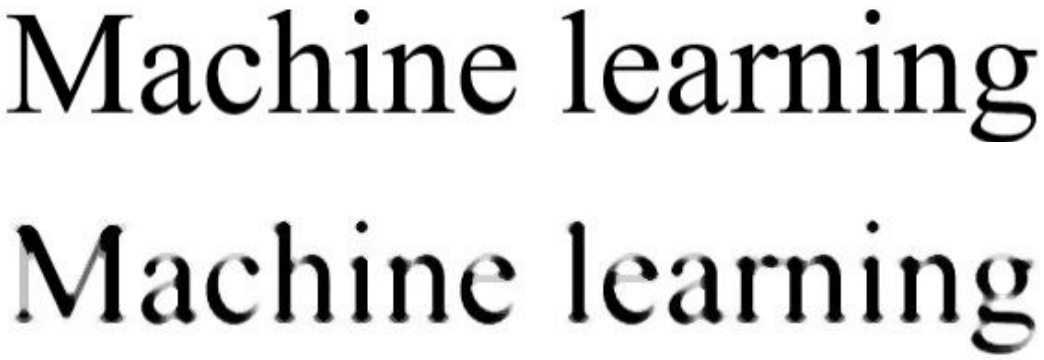
Obr. 12 – Obraz před erozí (vlevo) a po erozi (zdroj: vlastní)

Aplikace mediánového filtru nahradí každý pixel mediánem hodnot okolních pixelů (kolik pixelů se použije pro výpočet, je dáno velikostí jádra), čímž umožnuje odstranit zbývající nežádoucí šum z obrazu. Tento typ filtrů je obzvláště efektivní pro potlačení šumu typu Salt & Pepper. Pokud se na obraz A použije mediánový filtr s oknem velikosti K x K, pak aktualizovaná hodnota pixelu $A_{i,j}$ je dána následujícím vztahem (Gonzales, Woods, 2008).

$$A_{i,j} = median\{A_{i-m,j-m}, A_{i-m,j-m+1}, \dots, A_{i+m,j+m}\} \quad (11)$$

Kde:

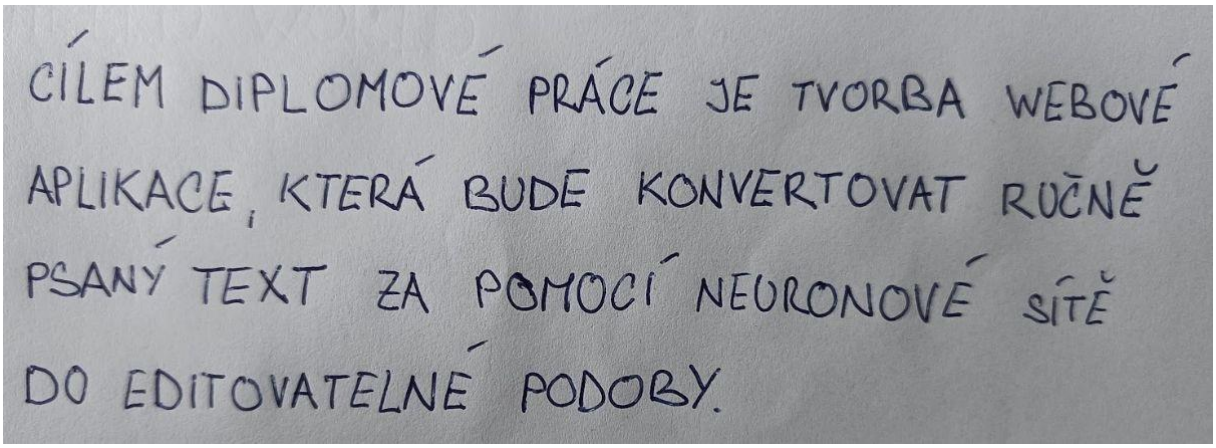
- $m = \left|\frac{K}{2}\right|$ (zaokrouhleno dolů), a tedy okno pokrývá sousední pixely od *i-m* do *i+m* a od *j-m* do *j+m*



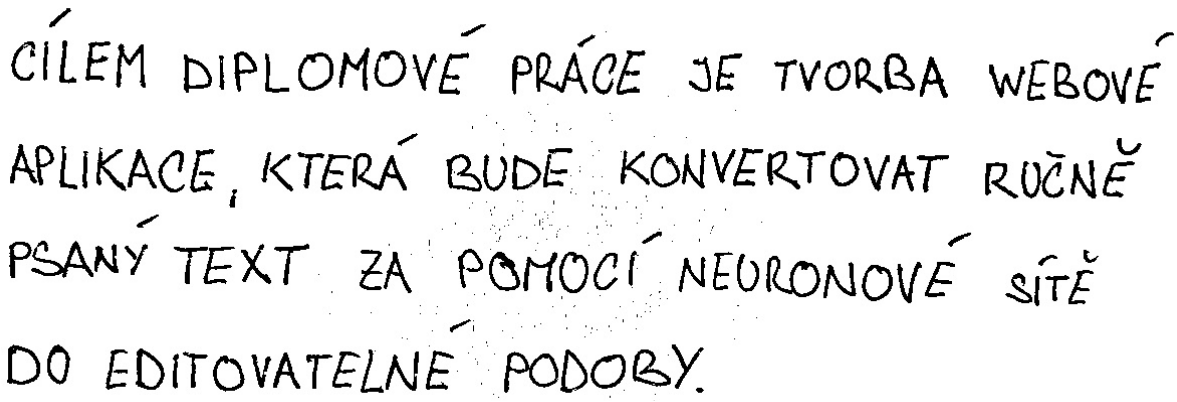
Obr. 13 - Ukázka aplikace mediánového filtru (zdroj: vlastní)

1.3.5 Limitace předzpracování obrazu

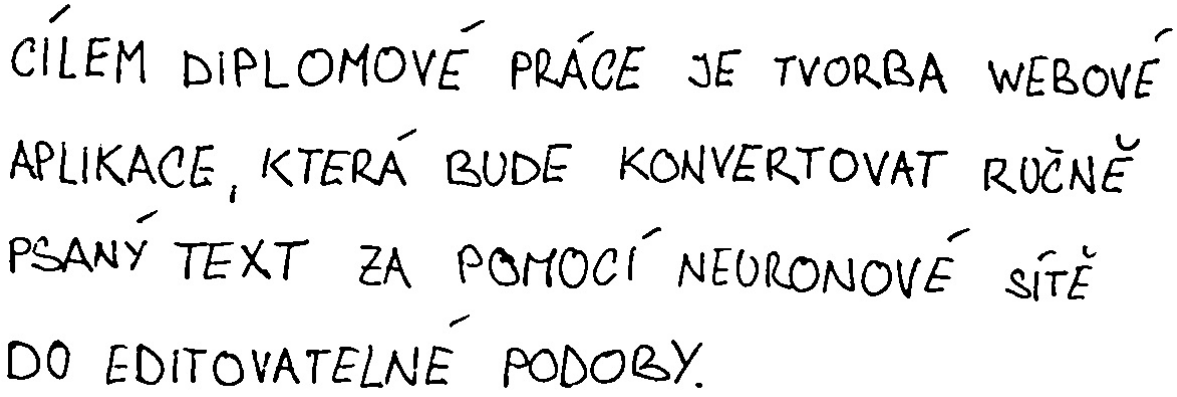
Zmíněné techniky předzpracování obrazu zahrnující grayscaling, binarizaci, ošetření rotace a odstranění šumu mají zásadní vliv na kvalitu OCR. Je ale potřeba zmínit, že jednotlivé parametry je potřeba volit velmi obezřetně a korektní nastavení parametrů jednotlivých metod vyžaduje značné množství nastavování metodou pokus-omyl. Je totiž možné, pokud se zvolí například příliš velký filtr u morfologických operací, že dojde k naprostému zdeformování písma. Bohužel neexistuje univerzální nastavení všech zmíněných metod. Atomatizované korektní nastavení parametrů binarizace a technik sloužících k odstranění šumu pro obraz s libovolnou kvalitou světelných podmínek, je téma, které by mohla pokrýt samostatná diplomová či disertační práce. Tato práce tak ve své praktické části využívá fixních hodnot, které byly na základě empirických dat zvoleny jako ideální k typu předkládaných vstupních dat. Z tohoto důvodu, jak bude zmíněno dále, je nutné vstupní data předkládat v rámci určitých pravidel. Možné výsledky předzpracování obrazu jsou zachyceny na následujících obrázcích.



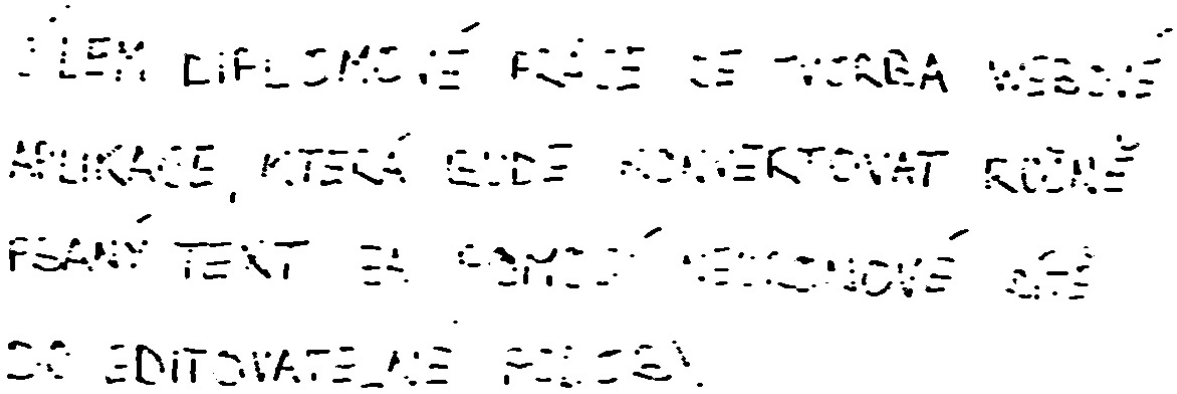
Obr. 14 - Obrázek před předzpracováním obrazu (zdroj: vlastní)



Obr. 15 - Obrázek po binarizaci obrazu, který obsahuje šum Salt & Pepper (zdroj: vlastní)



Obr. 16 - Binarizovaný obrázek s odstraněným šumem (korektní nastavení parametrů)



Obr. 17 - Binarizovaný obrázek s odstraněným šumem (špatné nastavení parametrů)

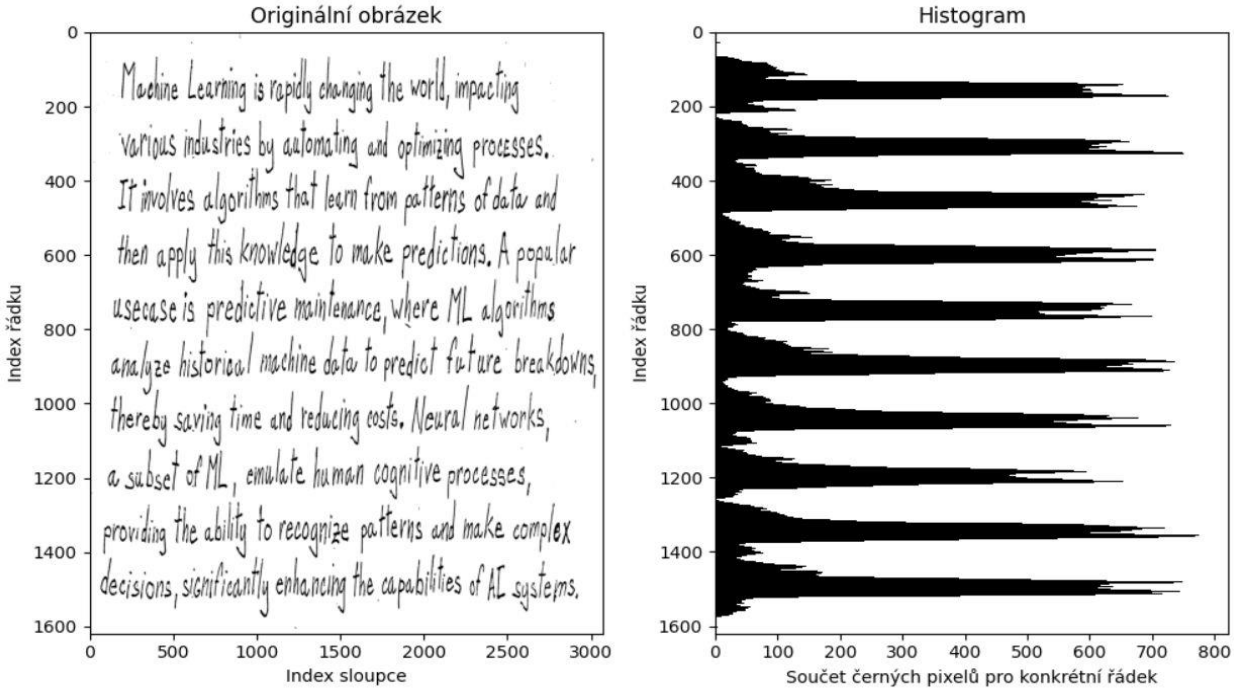
1.4 SEGMENTACE OBRAZU

Poté co je obraz úspěšně předzpracován, je možné ho rozdělit segmenty, nad kterými bude později vykonáváno samotné rozpoznávání. V tomto konkrétním případě segmenty představují řádky, na kterých se nacházejí jednotlivá písmena, která následně tvoří jednotlivé segmenty. Existuje několik možných způsobů, jak detekovat jednotlivé řádky, a sice projekční histogramová metoda, Houghova transformace, metoda detekce hran, algoritmus rozvodí („watershed algorithm“), algoritmus posuvného okna nebo využití modelu konvoluční neuronové sítě (Gonzales, Woods, 2008).

Následující vysvětlení principů se vztahuje pouze na ty metody, které byly v rámci práce otestovány.

1.4.1 Projekční histogramová metoda

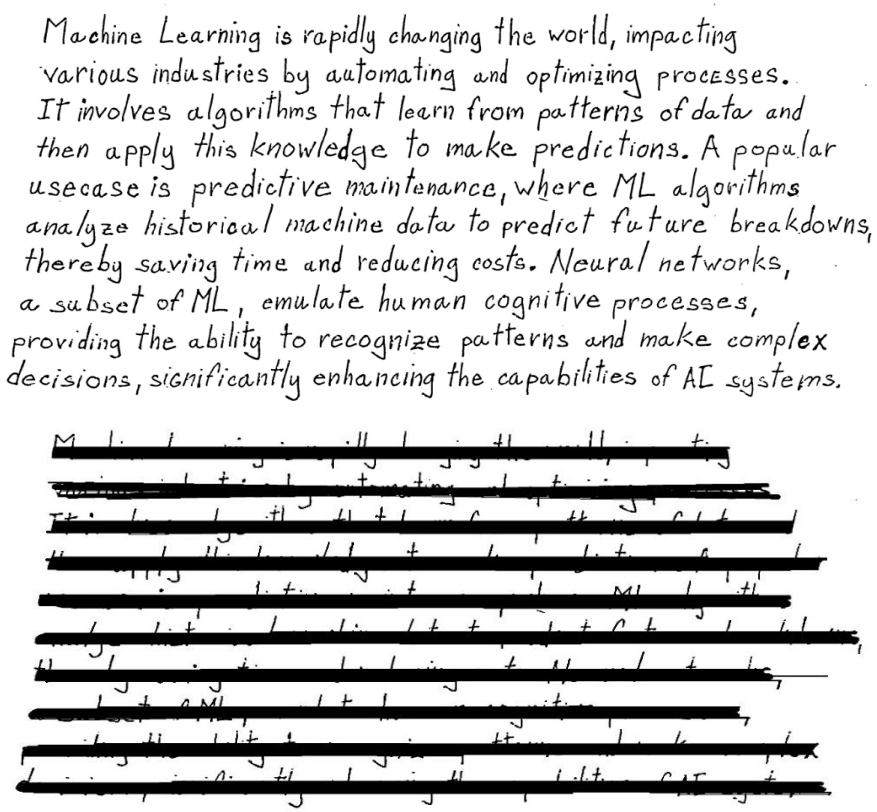
Projekční histogramová metoda pro detekci řádků funguje následujícím způsobem. Pokud má binarizovaný obrázek rozlišení 3000 x 1600 pixelů, pak je složen z 1600 řádků, kde každý řádek obsahuje 3000 pixelů. Pro každý řádek se spočítá počet černých pixelů (pokud je binarizovaný obrázek ve tvaru, kdy písmo v popředí je vyneseno černou barvou a pozadí bílou barvou). Poté je možné vytvořit graf, který na ose Y bude mít indexy jednotlivých řádků a na ose X množství černých pixelů pro individuální řádek (Gonzales, Woods, 2008).



Obr. 18 - Histogramová projekce řádků textu (zdroj: vlastní)

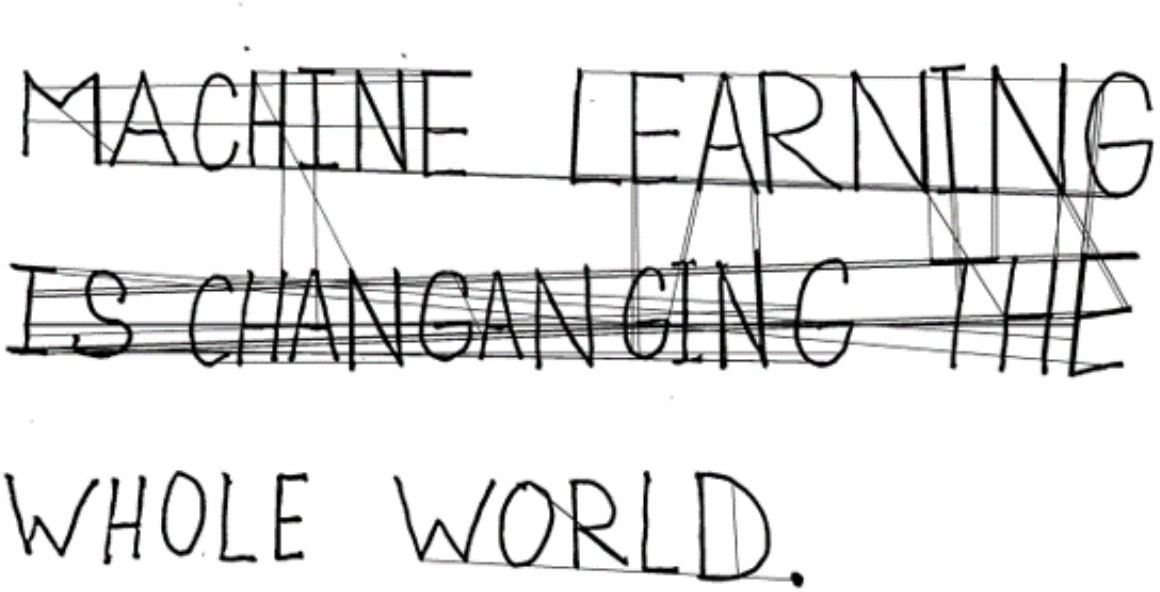
1.4.2 Houghova transformace

Další možností je již dříve zmíněna Houghova transformace, využívaná v aplikacích počítačového vidění k detekci přímek. Její současné modifikace rovněž umožnují detekci dalších geometrických tvarů. Principem této metody je, že každý bod v prostoru může reprezentovat čáru v jiném prostoru a naopak. Pokud je v obraze několik bodů, které mohou tvořit tvar (v tomto případě přímku), tak existují body v jiném prostoru, kde se tyto přímky protínají a tomuto prostoru se říká Houghův prostor. Pro existující obraz je vytvořen Houghův prostor, kde jsou všechny jednotlivé body v obraze popsány pomocí polárních souřadnic (ρ, θ). Když se poté setká několik bodů v Houghově prostoru v jednom bodě, tak to znamená, že v původním obraze existuje čára odpovídající tomuto bodu v Houghově prostoru. Parametr ρ v Houghově transformaci představuje vzdálenost od počátku souřadnicového systému (0, 0) k detekované přímce. Tato vzdálenost je měřena jako nejkratší délka přímky, která spojuje počátek souřadnicového systému s danou přímkou, a je kolmá na tuto přímku. Parametr θ je úhel (měřen v radiánech) mezi osou X a kolmicí na čáru, která je použita k změření parametru ρ (Kacmajor, 2017).



Obr. 19 - Aplikace Houghovy transformace na binarizovaný obrázek (zdroj: vlastní)

Jak je patrné z obrázku č. 18, tak s využitím metody projekčního histogramu nelze vždy spolehlivě určit výška řádku. Možným řešením je implementace komplexního algoritmu, který je schopný detekovat maxima a minima hodnot mezi jednotlivými řádky. Houghova transformace se jeví jako lepší varianta, ale v určitých případech dochází k detekci nežádoucích hran, viz následující obrázek.



Obr. 20 - Detekce nežádoucích přímek v obraze (zdroj: vlastní)

Tuto konkrétní problematiku lze řešit například ignorováním přímek, které s vodorovnou rovinou svírají větší úhel, než je předem stanovená hraniční hodnota.

1.4.3 Metoda posuvného okna

Jakmile je detekovaný řádek, resp. skupina řádků, je možné segmentovat jednotlivá písmena v rámci řádku. Metoda posuvného okna funguje na principu vytvoření obdélníku fixních rozměrů, který se následně posouvá po celé délce řádku (nebo obecně jakékoliv prohledávané oblasti) v daných intervalech a v každé iterace analyzuje obsah okna. Pro segmentaci obrazu v rámci OCR není tato metoda vhodná, protože nelze efektivně zajistit, že obsahem okna budou vždy jen jedno písmeno, což může vést ke špatné detekci (Rosebrock, 2015).

Obr. 21 - Princip algoritmu posuvného okna (zdroj: vlastní)

1.4.4 Metoda detekce hran

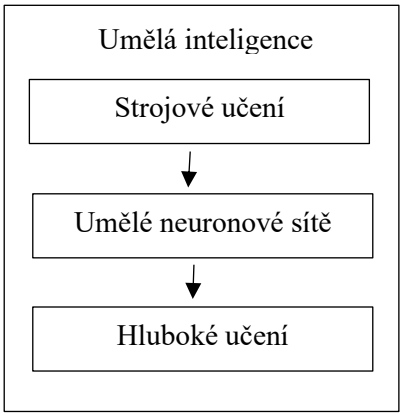
Více využívanou technikou je metoda detekce hran (kontur). Tato metoda (nebo spíše soubor metod) detekuje hrany v obraze tak, že identifikuje oblasti obrazu, kde dochází k náhlé změně intenzity (přechod mezi černou a bílou barvou). Konturou se poté rozumí křivka, propojující všechny kontinuální body, které mají stejnou barvu nebo intenzitu (OpenCV, 2023).

Jednou z možných implementací metody detekce kontur je algoritmus vytvořený dvojicí japonských inženýrů Satoshi Suzukim a Keiichi Abem. Algoritmus prochází obrázek po jednotlivých řádcích a pokud nalezne v řádku pixel, který je součástí objektu (bílý nebo černý pixel, záleží dle nastavení), pak je tento pixel brán jako startovní bod. Po nalezení tohoto pixelu, dojde k prohledání sousedních pixelů a hledání pokračuje směrem k pixelu stejné barvy. Hledání je ukončeno v moment, kdy algoritmus najde cestu k počátečnímu pixelu. V tento moment byla úspěšně nalezena kontura objektu. Algoritmus pokračuje dále v procházení obrázku, dokud nenalezne další pixel, který je součástí dalšího objektu (není součástí žádného z předchozích objektů), a postup se opakuje (Suzuki, Abe, 1985).

Obr. 22 - Detekce kontur v binarizovaném obrazu (zdroj: vlastní)

2 STROJOVÉ UČENÍ A NEURONOVÉ SÍTĚ V RÁMCI ICR

Strojové učení je součástí většího celku spadajícího pod odvětví informatiky známého jako umělá inteligence (AI neboli „Artificial Intelligence“). Zaměřuje se na zpracování dat pomocí metod a algoritmů, které imitují způsob řešení, kterým by daný problém řešil člověk. To umožňuje počítači v průběhu času zvyšovat svoji přesnost v řešení dané problematiky, jinými slovy, učit se. Jedná se o důležitou součást informatiky, která algoritmům díky statistickým metodám umožňuje učit se provádět klasifikace nebo předpovědi a odhalovat poznatky v projektech týkajících se datové vědy. Někdy se strojové učení mylně zaměňuje s pojmem „hluboké učení“ potažmo s pojmem „umělá neuronová síť“. Ač jsou všechny tři termíny podoblastmi umělé inteligence, existuje přesnější klasifikace, která stanovuje, že umělé neuronové sítě jsou podoblastí strojového učení a hluboké učení je podoblastí umělých neuronových sítí (IBM, 2023).



Obr. 23 - Hierarchie pojmů
(zdroj: vlastní)

Strojové a hluboké učení se odlišuje v rámci procesu učení algoritmu. Strojové učení vyžaduje lidský zásah, kdy člověk musí počítači poskytnout data ohledně vlastností, jaké má algoritmus použít, a také je potřeba data označit, aby algoritmus věděl, s jakými daty má pracovat. Hluboké učení sice může používat označené datové sady („labeled dataset“), ale není to nutnost, protože algoritmus je schopný přijímat nestrukturovaná data (text, obrázky) a určit sadu vlastností, které od sebe různé kategorie odlišují (IBM, 2023).

2.1 OBLASTI VYUŽITÍ NEURONOVÝCH SÍTÍ

Neuronové sítě je vhodné využít v případech, kdy řešení daného problému buď nelze popsat matematicky, aniž by došlo k vynechání nějaké klíčové informace či souvislosti, nebo v opačném případě, kdy matematický model řešení problému lze vytvořit, ale je příliš komplexní, než aby jeho následné řešení šlo zprostředkovat pomocí algoritmu. Mezi hlavní oblasti využití tak patří následující:

- Predikce časových řad a případné následné rozhodování (např.: vývoj akciového trhu, predikce rozvinutí nemocí u pacienta, předpověď počasí a mnoho dalších);
- Aplikace počítačového vidění (např.: rozpoznávání a rekonstrukce obrazu, rozpoznání ručně psaného textu a číslic);
- Využití v analýze signálů (např.: audio signály, video signály, signály ve zdravotnictví jako EEG a EKG, analýza rádio-frekvenčních signálů);
- Transformace signálů (převod psaného textu do Morseova kódu nebo mluvené řeči);
- Generování textu nebo obrazu (chatboti).

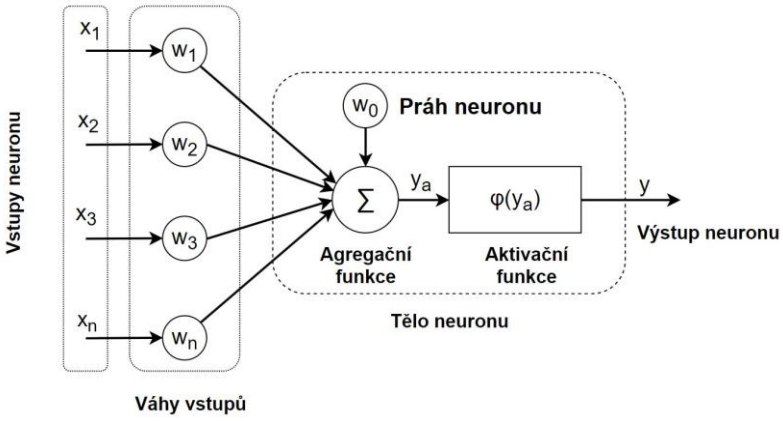
(Doležel, 2016).

2.2 STRUKTURA UMĚLÉHO NEURONU

Umělé neuronové sítě nebo jen zkráceně neuronové sítě mají základ v biologických neuronových sítích mozku. Jejich cílem je napodobování struktury a chování biologických neuronových sítí. Základním stavebním prvkem jak umělých, tak skutečných neuronových sítí je neuron, který představuje nervovou buňku (Doležel, 2016).

Komplexní porovnání mezi matematickým modelem neuronových sítí a umělým neuronem a jejich biologickým vzorem lze najít v publikaci „Úvod do umělých neuronových sítí“ (Doležel, 2016).

Každý neuron je propojen s dalšími neurony a má přidruženou váhu a práh. Pokud je výstup jakéhokoliv neuronu nad specifikovanou prahovou hodnotou, dochází k aktivaci toho neuronu a jeho výstup je předán na vstup další vrstvy. V opačném případě neuron není aktivován a žádná data nepředává (IBM, 2023).



Obr. 24 - McCullochův-Pittsův model neuronu (Doležel, 2016)

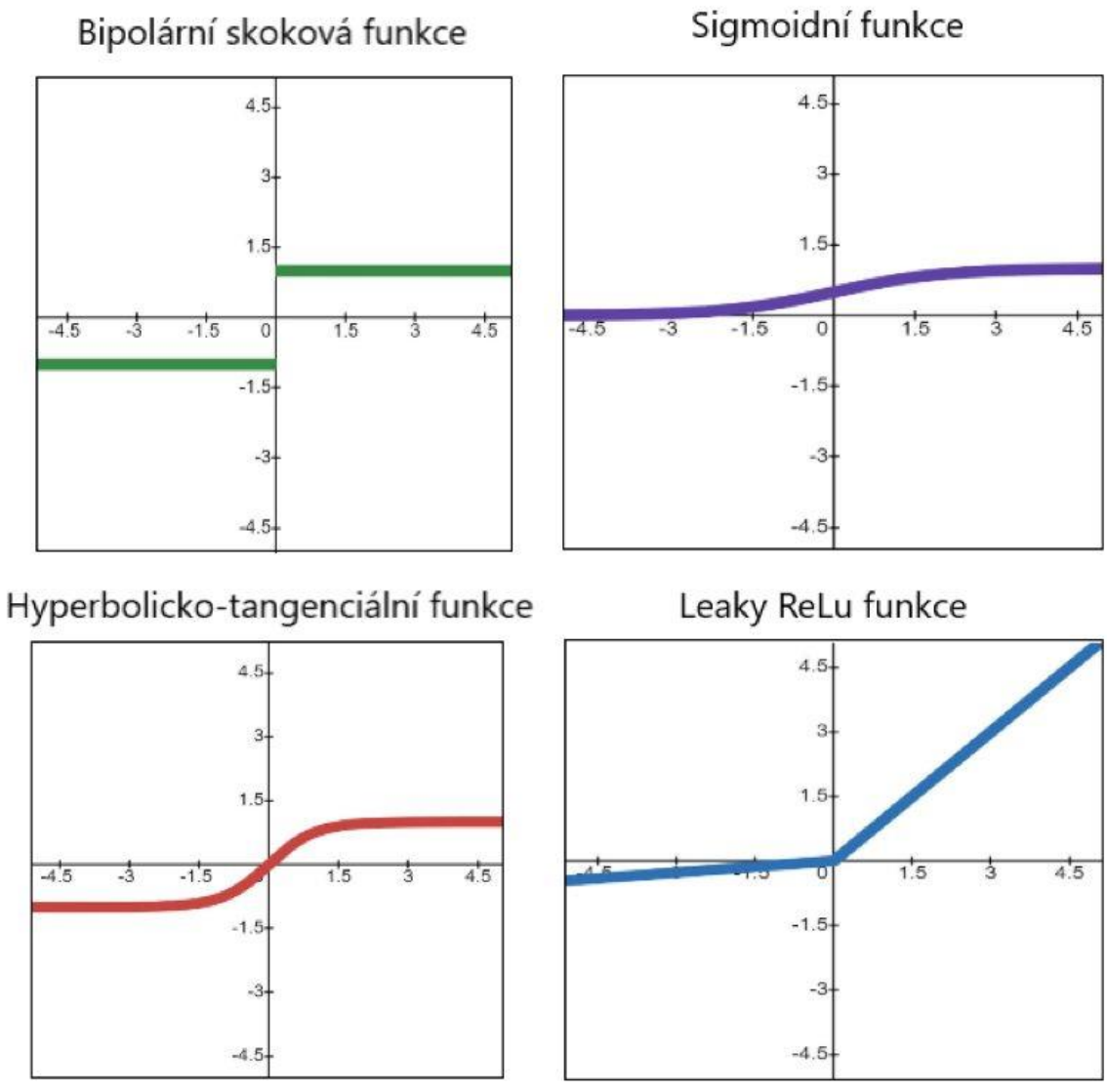
Obrázek č. 24 znázorňuje model neuronu, jak jej popsali Warren MuCulloh a Walter Pitts v roce 1943. Vstup do neuronu je označen jako vektor vstupů $\boldsymbol{X}$ (může se jednat o číselné hodnoty nebo hodnoty typu boolean). Váhy vstupů, které představují citlivost neuronu na daný vstup, jsou označeny jako vektor vah $\boldsymbol{W}$. Vztah mezi vstupem a jeho váhou lze matematicky popsat pomocí operátoru konfluence. Pro základní model neuronu se uvažuje operátor v linearizované podobě a konfluence je tak nahrazena součinem hodnot:

$$z_i = w_i \oplus x_i$$

$$z_i = w_i \cdot x_i$$

Váhy se následně agregují pomocí agregační funkce, která se využívá ke sloučení vektoru vstupních hodnot $\boldsymbol{X}$ potom co prošel operací konfluence s vektorem vah $\boldsymbol{W}$. Výsledkem by tak měl být skalární signál označovaný jako vstupní potenciál aktivační funkce y_a, který se

následně porovná s prahem neuronu w_0. Na základě výsledku tohoto porovnání buďto vstupní potenciál y_a vstupuje (nebo nevstupuje) do aktivační funkce. Aktivační funkce má za úkol převést tuto skalární hodnotu na výslednou výstupní hodnotu neuronu. Aktivačních funkcí existuje celá řada a dají se ze své podstaty rozdělit na čtyři kategorie: lineární, nelineární, spojité a diskrétní. Každý typ úlohy vyžaduje individuální zvážení, jaký typ aktivační funkce je pro danou úlohu vhodný (Doležel, 2016). Mezi nejčastěji používané se řadí následující: bipolární skoková funkce, sigmoidní funkce, hyperbolicko-tangenciální funkce, usměrněná lineární funkce (ReLu), děravá usměrněná lineární funkce (Leaky ReLu) (Baheti, 2021).



Obr. 25 - Aktivační funkce (zdroj: vlastní)

2.3 UČENÍ NEURONOVÝCH SÍTÍ

Schopnost učit se je pravděpodobně nejdůležitější vlastností neuronových sítí. Jedná se o proces, kdy síť mění a přizpůsobuje své nastavitelné parametry na základě tréninkových dat za účelem zlepšení své výkonosti v rámci řešení dané problematiky. Nejčastěji měnitelným parametrem jsou váhy a prahové hodnoty mezi jednotlivými neurony. Síť ale může například i upravovat vlastní strukturu nebo měnit strmost aktivační funkce. Jedná se o iterativní proces, který probíhá v tzv. epochách. Epocha označuje jeden průchod celého trénovacího souboru dat skrze trénovací algoritmus. Tyto algoritmy ovlivňují, jakým způsobem se budou parametry měnit. Kromě trénovacích algoritmů existují také algoritmy optimalizační, které určují jak rychle a efektivně se provádí učení podle konkrétního trénovacího algoritmu. Obecně existují dva základní přístupy učení, a to: učení s učitelem („supervised learning“) a učení bez učitele („unsupervised learning“) (Delua, 2021).

Data jsou typicky rozdělena buď do dvou (trénovací + testovací) nebo tří dávek (trénovací + testovací + validační) za účelem, aby vytrénovaný model mohl být nezávisle otestován bez zkreslených výsledků vzniklých přetrénováním. Trénovací data se používají pro nastavení vah a prahů a validační data se používají jako metrika toho, který model nejlépe generalizuje dosud neviděná data během trénování modelu, a pomocí této metriky upravuje hyperparametry modelu. Testovací data se využívají po dotrénování modelu jako konečné ohodnocení kvality modelu vzhledem k datům, na kterých byl model trénován (Brownlee, 2020). Samotné trénování poté může být ukončeno třemi způsoby. Buď je nastaven fixní počet epoch, po které dojde k ukončení trénování, nebo se sleduje hodnota chybové funkce a když klesne pod určitou hranici, tak dojde k ukončení trénování. Třetí a nejčastěji používaná varianta je nastavení vysokého počtu epoch s tím, že při dosažení nového minima chybové funkce dojde k uložení hodnot prahů a vah modelu v dané iteraci a model pokračuje v trénování. Jako výsledek se poté bere model z iterace, ve které dosáhl nejnižší chybové funkce. Učení s učitelem má tři varianty, a sice online, offline a dávkové. Při online variantě dochází k upravování parametrů průběžně s každou novou hodnotou, která je poskytnuta jako vstup do neuronové sítě. Offline varianta přírůstkové změny vah shromažďuje ve speciální proměnné a jsou aplikovány až potom co jsou všechny vzory trénovací množiny předloženy neuronové sítí. Dávková („batch“) varianta je kombinací online a offline metody, kdy se aktualizují váhy po n vzorcích (Doležel, 2016).

2.3.1 Učení s učitelem

Metoda učení s učitelem spočívá ve využití oštítkovaných datových sad („labeled datasets“). Pro každý vstup má tak síť informaci o tom, jaký by měl být očekávaný výstup. Tato metoda se typicky využívá v aplikacích klasifikace a regrese (rozeznání objektů, odhad ceny nemovitostí na základě vybavení domu atd.). Z matematického hlediska je tak cílem nalezení minima chybové funkce („loss function“). Tato funkce tak popisuje účinnost učení daného modelu neuronové sítě. Níže je uvedeno několik nejužívanějších chybových funkcí (Doležel, 2016).

2.3.1.1 Střední kvadratická chyba („Mean Squarred Error“)

Pravděpodobně nejjednodušší způsob výpočtu chybové funkce je střední kvadratická chyba. Využívá se v regresních problémech a je průměrem kvadratických rozdílů mezi předpovědí a skutečnou hodnotou (Whitfield, 2023).

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{12}$$

Kde:

- N je počet testovaných vzorků;
- y_i je předpovídaná hodnota;
- $\hat{y}_i$ je skutečná hodnota.

2.3.1.2 Chyba absolutního rozdílu („Mean Absolute Error“)

Chyba absolutního rozdílu se je obdobou střední kvadratické funkce. Liší se tím, že rozdíl předpovídané a skutečné hodnoty není brán jako druhá mocnina, ale jako absolutní hodnota (Whitfield, 2023).

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{13}$$

2.3.1.3 Křížová entropie („Cross-entropy Loss“)

Křížová entropie je nejvyužívanější funkce pro klasifikační úlohy. Hodnota chybové funkce roste, jak se předvídaná pravděpodobnostní distribuce vzdaluje od skutečné

pravděpodobnostní distribuce. Je nutné zmínit, že tato verze se využívá jen pro úlohy binární klasifikace. To znamená, že síť je schopna klasifikovat jen v rámci ano/ne (Whitfield, 2023).

$$CEL = -(y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)) \tag{14}$$

2.3.1.4 Kategorická křížová entropie („Cathegorical cross-entropy Loss“)

Kategorická křížová entropie se využívá pro mnohonásobnou třídní klasifikaci. Výsledná proměnná je tzv. „one-hot“ kódována. To znamená, že pokud je cílem úlohy klasifikace ručně psaných písmen, tak výsledek bude vektor 26 hodnot, kde 1 bude na místě odpovídajícím skutečné třídě a na všech ostatních místech bude hodnota 0. Jedná se o kombinaci aktivační funkce „softmax“ a křížové entropie. Aktivační funkce softmax transformuje výstupní skóre jednotlivých tříd na pravděpodobnostní distribuci (Gómez, 2018).

2.3.2 Učení bez učitele

Učení bez učitele nevyužívá oštítkovaných datových sad. Místo toho se využívá algoritmů, které umožňují analyzovat a seskupovat data s podobnými vlastnostmi. Síť je tak schopna detekovat skryté vzory bez lidského zásahu. Tohoto přístupu se využívá například při dolování dat, kdy se seskupují vydolovaná data na základě jejich podobností nebo rozdílů. Další možností je hledání asociací. Například jestli určitá skupina lidí, co kupuje produkt A, bude také kupovat produkt B. Typicky se tento styl učení uplatňuje u asociativních pamětí a Kohenenovy sítě (Doležel, 2016).

2.3.3 Trénovací algoritmy

Jedním ze základních algoritmů je Zpětné šíření chyby („Backpropagation“). Pro lepší pochopení tohoto algoritmu je potřeba definovat, co je to gradientní sestup a řetízkové pravidlo derivací („chain rule“). Gradientní sestup („gradient descent“) je základní matematický optimalizační algoritmus. V kontextu neuronových sítí je gradient směr, kterým se algoritmus posouvá ve snaze minimalizovat chybovou funkci. Jeho cílem je nalezení co možná nejlepších parametrů modelu, které minimalizují chybovou funkci. Tento algoritmus pracuje iterativně, to znamená, že v každém kroku upravuje parametry modelu ve směru, který nejvíce snižuje chybu. Takový směr se označuje jako směr negativního gradientu chybové funkce. „Sestup“ poté označuje fakt, že se metoda pokouší „sestoupit“ k nejnižšímu bodu chybové funkce, který odpovídá nejlepším parametrům modelu. Existuje několik variant tohoto algoritmu, kde je

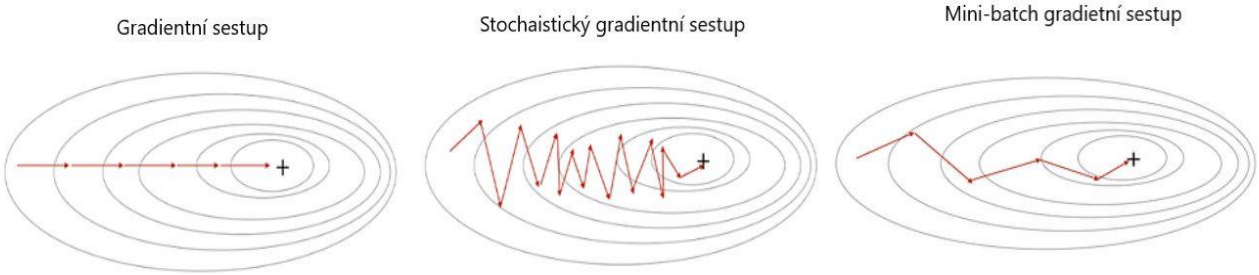
možné uvést např.: dávkový gradientní sestup, stochastický gradientní sestup nebo mini-dávkový gradientní sestup. Řetízkové pravidlo je základní metoda pro výpočet derivace složené funkce. V kontextu neuronových sítí je velmi užitečná pro efektivní výpočet gradientu chybové funkce přes všechny váhy a prahy v síti. Algoritmus Zpětného šíření chyby pak kombinuje algoritmus gradientního sestupu a řetízkové pravidlo do efektivní metody pro trénování neuronových sítí. Pomocí dopředného průchodu se vypočte chyba na výstupu a poté se pomocí zpětného průchodu efektivně spočítá, jak by se chyba změnila při malé změně každé z vah. Tyto „derivace chyby“ jsou pak použity v gradientním sestupu k malé úpravě vah v síti tak, aby se celková chyby snížila. Je důležité zmínit, že aktivační funkce musí být derivovatelná (Kostadinov, 2019).

Dalším používaným algoritmem, který je možno zmínit, je Hebův zákon učení. Tento algoritmus vychází z nervových systémů organismů a stanovuje, že pokud se dva neurony aktivují společně, pak se jejich váha posílí. V opačném případě se spojení mezi nimi oslabí. Typicky se tento přístup uplatňuje u sítí typu jednoduchý perceptron nebo Hopfieldovy sítě (Doležel, 2016).

2.3.4 Optimalizační algoritmy

Jak bylo zmíněno v kapitole 2.3 , optimalizační algoritmy slouží k dosažení stejného cíle jako trénovací algoritmy, ale působí na jiné úrovni procesu učení. Zatímco trénovací algoritmy definují celkový rámec, jakým způsobem se bude učení uskutečňovat (jak se budou upravovat váhy a prahové hodnoty na základě vstupních dat a hodnoty chybové funkce), optimalizační algoritmy jsou konkrétní techniky, které se používají k provedení těchto úprav.

Jedním ze základních optimalizačních algoritmů je v předchozí kapitole zmíněný gradientní sestup a jeho modifikace.



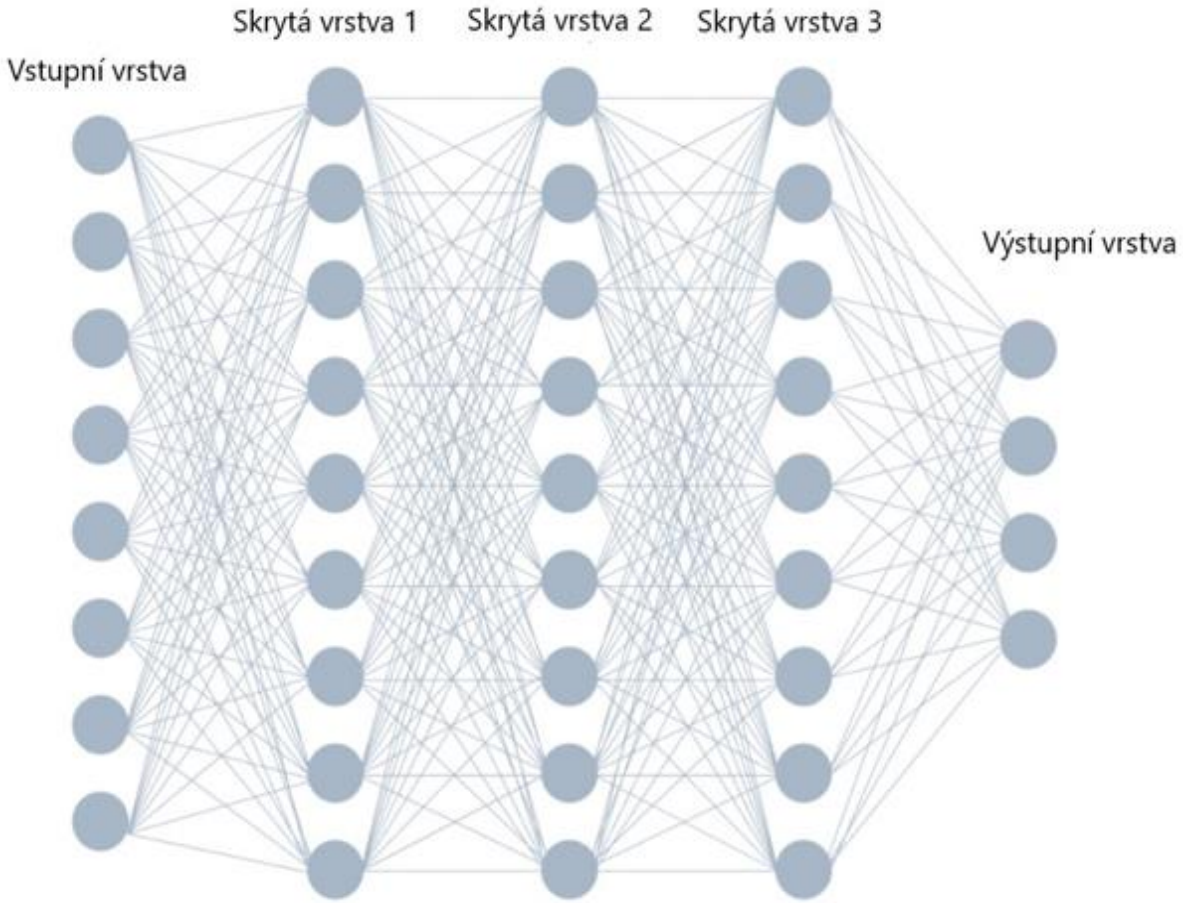
Obr. 26 - Vizualizace gradientního sestupu pro různé metody (Chauhan, 2020)

Dalším velmi často užívaným algoritmem je Adam („Adaptive Moment Estimation“). Adam využívá koncepci momentu tím, že kombinuje prvky z minulých gradientů k tomu, aby určil, jakým směrem se aktuálně posunout. Kombinuje algoritmy stochastického gradientního

sestupu (SGD) a RMSprop („Root Mean Square Propagation“). RMSprop algoritmus se snaží řešit problém, kdy se učení v některých dimenzích děje rychleji než v jiných, což může mít za následek problémy s konvergencí. Tento problém se snaží řešit adaptivní úpravou rychlosti učení pro jednotlivé parametry. Existuje množství variant tohoto algoritmu, např.: AdaGrad, AdaDelta, AdamW, NAdam (Chauhan, 2020).

2.4 TYPY VRSTEV NEURONOVÝCH SÍTÍ

Neuronové sítě jsou tvořeny vrstvami neuronů. Základní struktura umělé neuronové sítě se skládá ze vstupní vrstvy, jedné nebo více vnitřních (skrytých, hlubokých) vrstev a výstupní vrstvy. „Hluboké“ v názvu hluboké učení tak odkazuje na počet skrytých vrstev neuronové sítě. Pokud se struktura sestává z více než tří vrstev (vstupní, n skrytých vrstev, výstupní), pak hovoříme o hlubokém učení, potažmo hluboké neuronové síti (IBM, 2023).



Obr. 27 - Model plně propojené hluboké neuronové sítě (Baheti, 2021)

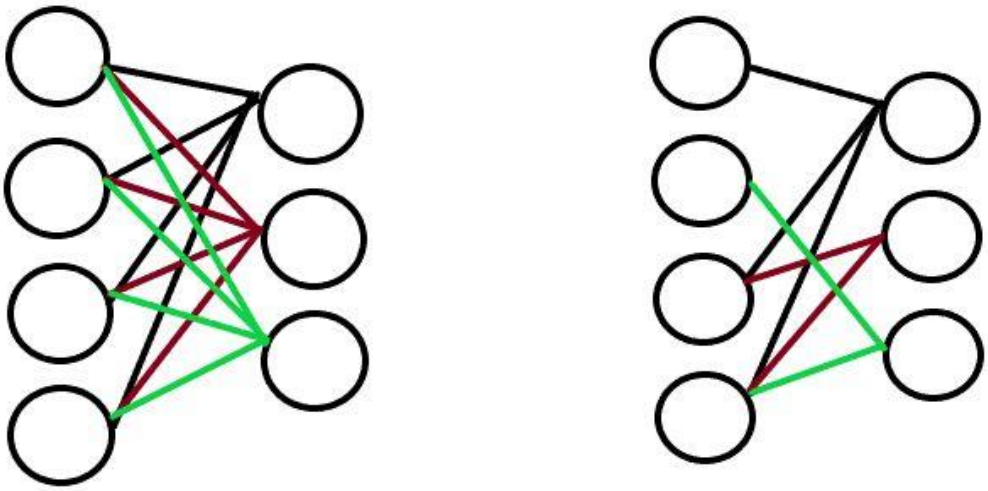
V oblasti architektury umělých neuronových se lze setkat s širokou škálou různých vrstev, z nichž každá má svou typickou aplikaci. Dále je uvedeno a podrobněji popsáno několik typů vrstev, které jsou nejčastěji používány v rámci řešení problematiky OCR pomocí neuronových sítí.

2.4.1 Vrstva Dense

Vrstva Dense mezi sebou propojuje všechny neurony mezi dvěma sousedními vrstvami. Tento typ je typický pro dopředné neuronové sítě, ale využívají jej například i konvoluční nebo rekurentní neuronové sítě. V topologii sítě se tento typ vrstev vždy využívá jako poslední. Slouží ke klasifikaci nebo regresi na základě vlastností detekovaných v předchozích vrstvách sítě. V případě úlohy klasifikace písmen bude mít plně propojená vrstva 26 neuronů, kde každý neuron odpovídá písmenu abecedy. Ukázka Dense vrstev je na obrázku č. 27 (Rosebrock, 2021).

2.4.2 Vrstva Drop-out

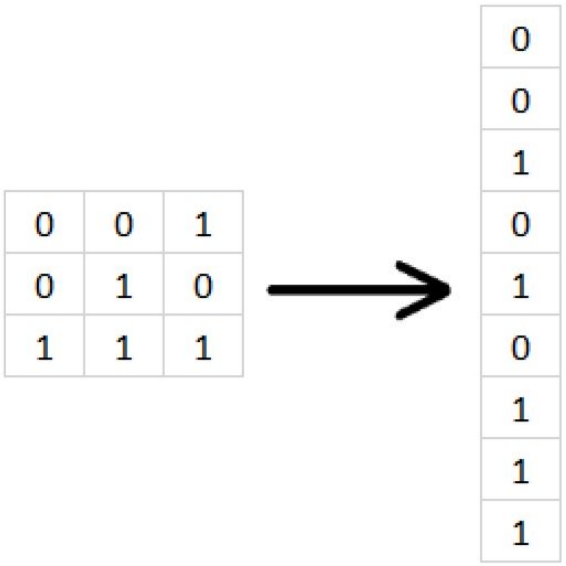
Drop-out vrstva funguje jako regulační vrstva, která funguje opačně než vrstva Dense a sice tak, že definuje procentuální množství neuronů v dané vrstvě, které nebudou aktivovány. Hlavním smyslem této vrstvy je prevence přeučení sítě („overfitting“). Přeučení sítě je situace, kdy se model příliš přizpůsobí trénovacím datům a není kvůli tomu schopen správně klasifikovat data, která zatím neviděl. V takovém případě se o daném modelu sítě říká, že není schopen data generalizovat (Rosebrock, 2021).



Obr. 28 - Rozdíl mezi vrstvou Dense (vlevo) a Drop-out (zdroj: vlastní)

2.4.3 Vrstva Flatten

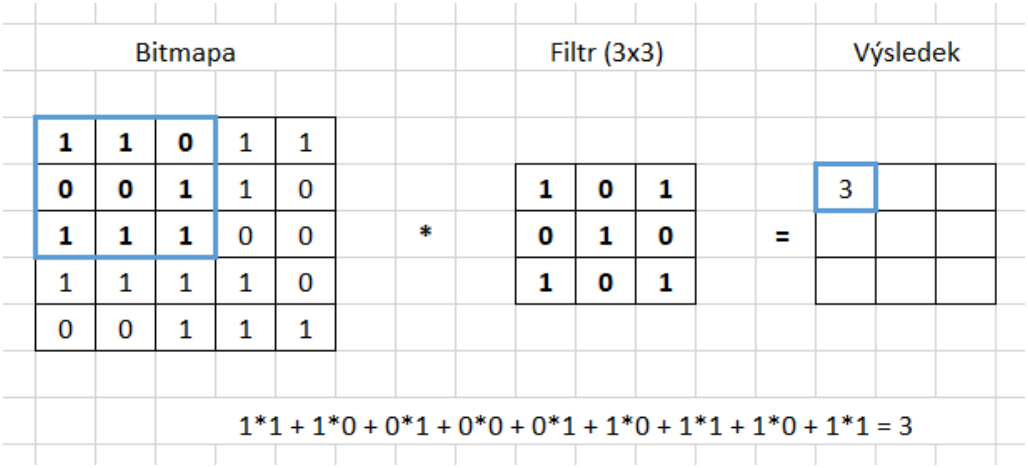
Vrstva Flatten se využívá k transformaci vstupních dat (například výstup z konvoluční vrstvy) ve formě N-rozměrného pole na vektor. Jelikož plně propojená vrstva na výstupu očekává na svém vstupu data v jednorozměrném formátu, tak se tyto dvě vrstvy velmi často používají spolu (McLean, 2021).



Obr. 29 - Princip vrstvy Flatten (zdroj: vlastní)

2.4.4 Konvoluční vrstva

Konvoluční vrstvy jsou základní stavební bloky pro konvoluční neuronové sítě, které jsou typicky nejčastěji používány v aplikacích počítačového vidění. Využívají operace konvoluce mezi obrazem a konvolučním jádrem. Jádro, které je možné si představit jako matici celých čísel, která typický bývá čtvercová (ale nemusí být), funguje jako filtr. Filtr se posouvá po vstupním obrazu a provádí konvoluci prvků filtru s odpovídajícími prvky vstupních dat obrazu. Výsledkem je sada map příznaků z anglického originálu „feature maps“, které zachycují různé aspekty vstupních dat. Na obraz se může aplikovat řada filtrů, kde každý bude mít jiný účel např.: jeden filtr detekuje určité barvy, další detekuje hrany atd. Mapy vlastností tak popisují, jak silně se daná vlastnost objevuje v různých místech vstupních dat. V rámci konvolučních neuronových sítí jsou to právě hodnoty prvků filtru, které se učí pomocí metody zpětného šíření chyby (Rosebrock, 2021).



Obr. 30 - Ukázka aplikace konvolučního filtru na vstupní bitmapu

Jak lze vidět na obrázku č. 30, výsledkem aplikování filtru je zmenšení výsledného obrazu. Pokud má obrázek rozměry N x N a filtr má rozměry U x U, tak výsledné rozměry obrázku budou (N-U+1) · (N-U+1). Tomuto výsledku se dá předejít několika způsoby, ale nejpoužívanějším řešením je přidání okrajových nulových hodnot do vstupního obrazu. Díky tomu nedojde ve výsledku ke zmenšení obrazu. Další možností je nastavení parametru stride (parametr udávající o kolik pixelů se filtr posune po každé aplikaci) na hodnotu 1. (Reynolds, 2019).

2.4.5 Vrstva Pooling

Pooling vrstva se využívá ke zmenšení rozměrů matice, čímž se snižuje výpočetní náročnost při trénování sítě. Zároveň jejím využitím dochází ke zmenšení šance na přeučení sítě, protože dochází ke zmenšení množství informací, avšak ne v takové míře, aby se snížila přesnost. Stejně jako konvoluční vrstva, tato vrstva také využívá filtr, který se systematicky posouvá po obraze. Tento filtr, ale nemá žádné váhy a pouze vymezuje na jakou část vstupních dat se použije agregační funkce. Existují různé varianty Pooling vrstev definované právě typem použité agregační funkce. Existuje například Max-pooling varianta, která využívá výběru největší hodnoty z pole označeného filtrem, nebo Average-pooling varianta, která vypočítá a vybere aritmetický průměr z hodnot v poli. Takto vybrané hodnoty se poté umísťují na výstup (IBM, 2023).

2.5 TYPY ARCHITEKTUR NEURONOVÝCH SÍTÍ

Architektura neuronových sítí je udávána množstvím parametrů: typ sítě, počet vrstev, způsob zapojení vrstev za sebe, počet neuronů v jednotlivých vrstvách, konkrétní aktivační funkcí a v neposlední řadě také trénovací a optimalizační algoritmus. Níže je uvedeno několik základních typů neuronových sítí.

2.5.1 Perceptron

Perceptron je nejjednodušší model neuronové sítě, který se skládá z jediného neuronu (obrázek č. 24). Využívá učení s učitelem (je možné využít i Hebbovo učení) a používá se v binárních klasifikačních úlohách, to znamená tam, kde lze výsledná data rozdělit do dvou skupin. Výsledná data však musí být lineárně rozdělitelná rovinou, čehož lze dosáhnout využitím lineárně vážené agregační funkce. Matematicky lze výstup perceptronu popsat následovně:

$$y = \varphi(y_a) \tag{15}$$

Kde:

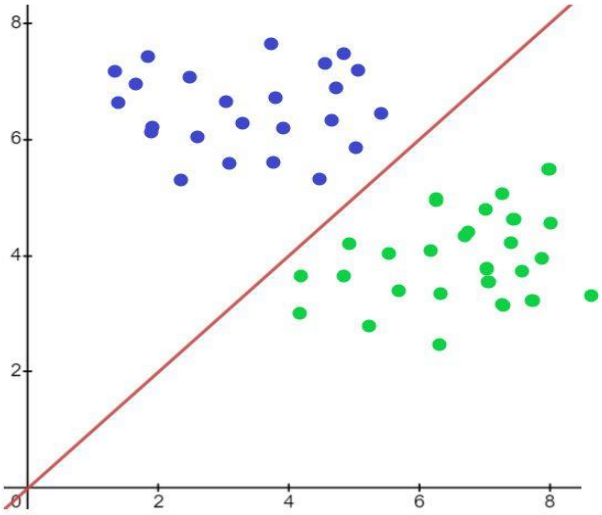
- φ představuje lineární aktivační funkci;
- y_a je potenciál aktivační funkce, který lze vyjádřit následovně:

$$y_a = \sum_{i=1}^{R} x_i \cdot w_i + w_0 \tag{16}$$

Kde:

- R je počet vstupů do perceptronu;
- x_i je vstup;
- w_i je váha vstupu;
- w_0 je práh neuronu.

(Doležel, 2016).



Obr. 31 - Výsledek binární klasifikační úlohy
(zdroj: vlastní)

2.5.2 Dopředná vícevrstvá neuronová síť („Multi Layer Perceptron“)

Dopředná vícevrstvá síť (DVS), někdy též označována jako vícevrstvý perceptron je základní topologií hlubokých neuronových sítí. Jedná se již o dříve zmíněný model plně propojené hluboké neuronové sítě ilustrovaný na obrázku č. 27, který se skládá ze vstupní vrstvy, skrytých vrstev a výstupní vrstvy. Počet skrytých vrstev typicky záleží na složitosti a povaze dané úlohy. Konkrétní počet je ve výsledku většinou určen na základě empirických dat a pozorování. Obecně lze buď ze začátku nastavit vyšší počet skrytých vrstev a postupně ubírat nebo využít jednodušší topologii a postupně ji zesložiťovat (Doležel, 2016). Charakteristickým znakem této sítě je, že k přenosu informace dochází pouze dopředným směrem, není zde žádné zpětné propojení neuronů nebo propojení neuronů v rámci jediné vrstvy. Stejně jako jednoduchý Perceptron využívá tento typ sítě učení s učitelem a algoritmus Zpětného šíření chyby, popsaný v kapitole 2.3.3 . Tuto síť lze využít pro široké spektrum úloh zahrnující: komplexní klasifikace, regresní úlohy (předpověď kontinuálního výstupu), aproximace funkcí, zpracování obrazu a zvuku a další (Bento, 2021).

2.5.3 Rekurentní neuronová síť

Rekurentní neuronové sítě (RNN), na rozdíl od standartních dopředných neuronových sítí, obsahují cyklické spoje mezi neurony, které umožňují informaci cirkulovat ve smyčce. To umožnuje sítí pamatovat si předchozí informace a použít je při výpočtech pro následující prvky v posloupnosti. Primárně se tak tento typ sítí využívá pro práci s posloupnostmi dat, jako jsou

časové řady nebo text. Hlavním problémem tohoto typu sítí je tzv. problém mizejícího gradientu („vanishing gradient problem“). Ten způsobuje, že při trénování sítě se gradienty, které určují, jak se upravují váhy v průběhu trénování mohou stát velmi malými, což vede ke zpomalení procesu učení. Čím delší je sekvence dat, tím znatelnější je tento problém. Z tohoto důvodu byly vytvořeny speciální typy rekurentních sítí jako je LSTM („Long Short Term Memory“) nebo GRU („Gated Recurrent Unit“), které jsou navrženy tak, aby tento problém minimalizovali a umožnily tak efektivnější trénování. Primární využití nacházejí rekurentní sítě například v aplikacích rozpoznání řeči, generování textu („Natural Language Processing“), strojový překlad nebo predikce dat časových řad (A. Amidi, S. Amidi, 2019).

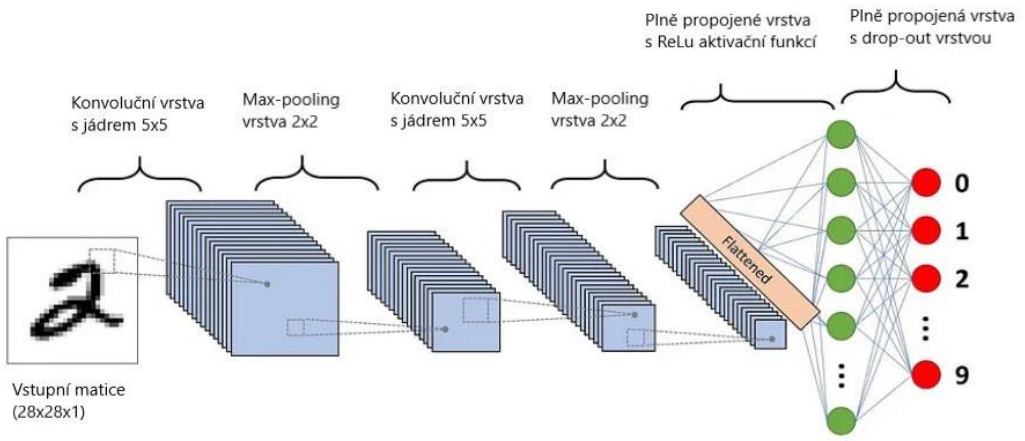
Spadá sem například Hopfieldova síť, která se používá jako autoasociativní paměť k rozpoznávání vzorů. Obecně mají klasifikační úlohy dvě fáze. V první fázi se síť učí jednotlivé vzory, v momentě, kdy je proces učení ukončen, přichází na řadu fáze vybavování, kdy se síť snaží klasifikovat daný vzor na svém vstupu. V případě, že síť na vstupu obdrží vzor, který není schopná zařadit, tak mohou nastat 3 různé situace:

- Fantomová detekce – sít konverguje ke vzoru, který nebyl součástí paměti;
- Síť nekonverguje, ale osciluje mezi dvěma stavy (méně pravděpodobné);
- Síť konverguje k nějakému vzoru uloženému v paměti, který ale neodpovídá vzoru na vstupu (velmi časté).

Pro komplexnější klasifikační úlohy jako rozeznání ručně psaného písma, tak není Hopfieldova síť zcela vhodná, kvůli nízké kapacitě paměti a nejistotě s jakou může udávat výsledek (Doležel, 2016).

2.5.4 Konvoluční neuronová síť

Konvoluční neuronová síť, jak již bylo uvedeno v kapitole 2.4.4 využívá ve své topologii konvoluční vrstvy. To jí umožnuje efektivně pracovat s daty, která mají mřížkovou topologii, jako je například obraz (barevný i černobílý). Primární uplatnění tak nacházejí při komplexních klasifikačních úlohách (detekce objektů, rozpoznávání vzorů, segmentace obrazu) nebo analýzy video a audio signálů. Nicméně aby konvoluční sítě byly schopné efektivně zpracovávat obraz potažmo video a audio signál, musí obsahovat i další vrstvy, které byly rovněž rozebrány v kapitole 2.4 . Topologie konvoluční neuronové sítě sloužící k rozeznání ručně psaných číslic by tak mohla mít strukturu, jakou popisuje obrázek č. 32.



Obr. 32 - Ukázka konvoluční neuronové sítě pro klasifikaci ručně psaných číslic (převzato a přeloženo z (Saha, 2018))

Pro topologii uvedenou na obrázku č. 32 je průběh zpracování vstupní matice dat následující. První konvoluční vrstva provede konvoluci vstupních dat s jádrem o rozměrech 5x5. Tím dojde k vytvoření nové matice příznaků, která představuje výstup této vrstvy. Tato matice je potom vložena na vstup první podvzorkovací vrstvy s jádrem o rozměrech 2x2, která má za cíl zredukovat velikost dat a zároveň zachovat důležité informace. Tyto první dvě operace se následně zopakují. Výstup z druhé podvzorkovací vrstvy je potom přiveden na vstup serializační vrstvy, která převede matici na vektor. Tento vektor je následně přiveden na vstup plně propojené vrstvy, která tato data zpracuje pomocí vah a prahových hodnot a na výsledek aplikuje ReLu aktivační funkci. Druhá plně propojená vrstva má poté stejný význam, ale ještě je spojena s vypouštěcí vrstvou, která během trénovací fáze zajišťuje, že nedojde k přeučení sítě.

2.5.5 Transformační neuronová sít

Transformační neuronová síť je model, který je schopný učit se kontext (v rámci dané úlohy) detekcí a sledováním vztahů v sekvenčních datech, jako jsou slova ve větě. Tento typ neuronových sítí byl poprvé představen roku 2017 v publikaci „Attention Is All You Need“ (Vaswani, 2017) skupinou vývojářů z firmy Google. Tento model představuje převratnou alternativu k tradičním „sequence-to-sequence“ architekturám založených na rekurentních neuronových sítích. Od těchto architektur se tento model liší tím, že není závislý na sekvenci dat. Díky tomu je schopen paralelně zpracovávat data, což má za následek kratší dobu trénování. Hlavní vlastností tohoto modelu je mechanismus pozornosti, který je označován jako „multi-head attention“, což umožňuje modelu se zaměřit na různé části vstupu nezávisle a poté kombinovat tyto informace do finální podoby. Jedná se o model, který využívá učení bez učitele a nepotřebuje tak, aby datové sady byly oštítkované (Merritt, 2022).

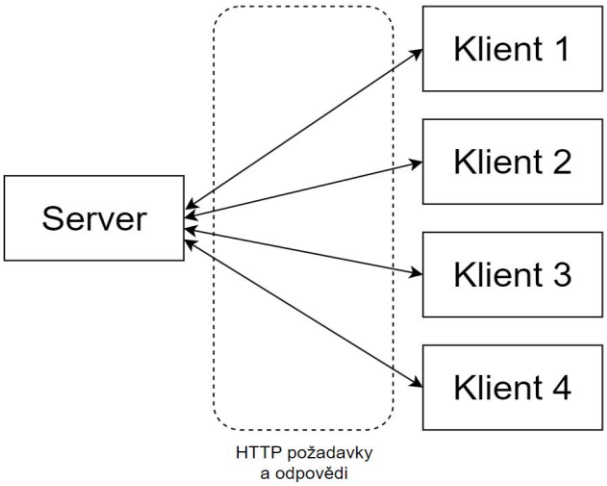
Transformační modely v současné době naprosto dominují celému odvětví umělé inteligence, a to převážně díky velkým jazykovým modelům („large language models“) a zpracování přirozeného jazyka („natural language processing“).

Asi nejznámějším současným zástupcem velkých jazykových modelů je GPT-4 („Generative Pretrained Transformer 4“) od firmy OpenAI nebo BERT („Biderectional Encoder Representations from Transformers“) od firmy Google. Tyto modely transformační sítě jsou navrženy a vytrénovány k tomu, aby dokázaly generovat lidský text. Fungují na základě statistické analýzy, kdy zpracovávají obrovské množství textu a učí se na těchto datech předvídat, jaká slova nejpravděpodobněji následují po jiných slovech. Díky tomu vzniká dojem, že model chápe kontext lidských vět a je schopen na ně reagovat stejným způsobem jako by reagoval člověk. Zpracování přirozeného jazyka je obor informatiky a umělé inteligence, který se zabývá právě interakcí mezi počítačem a lidským jazykem. Tento obor se snaží naučit počítače porozumět, interpretovat a generovat lidský jazyk tak, jak to dělají lidé (Merritt, 2022).

Obecně je tento druh sítí teprve na vzestupu a dá se očekávat, že v následujících letech bude udávat směr, kterým se bude vývoj umělé inteligence vyvíjet. Už v současné době se osvědčil v celé řadě aplikací: strojový překlad, generování textu, porozumění textu, převedení textu na řeč v reálném čase a mnoho dalších.

3 NÁVRH A REALIZACE SERVEROVÉ ČÁSTI APLIKACE

Aplikace využívá topologie server-klient, kde je serverová část zodpovědná za výpočty a práci s daty a klientská část aplikace představuje webové rozhraní, se kterým interaguje uživatel. Klient může být obecně jakékoliv zařízení/aplikace, která umí posílat http požadavky. Server je poté zodpovědný za odesílání odpovědí na tyto požadavky. Jejich vzájemná komunikace a sdílení dat je zprostředkována skrze REST API („Representational State Transfer Application Programming Interface“). Jedná se o architektonický styl, který využívá standartní metody a protokoly http pro komunikaci mezi klientem a serverem. Základní vlastností REST API je bezstavovost, to znamená, že každý požadavek musí obsahovat všechny informace potřebné k provedení dané operace. Využívají se základní http metody jako: GET (čtení), POST (vytvoření), PUT (aktualizace), DELETE (smazání) a další (Redhat, 2020).



Obr. 33 - Topologie server-klient (zdroj: vlastní)

Data se sdílejí buď ve formátu JSON („Javascript Object Notation“) nebo XML („Extensible Markup Language“). Topologie server-klient spolu s využitím REST API je populární volbou ve tvorbě webových aplikací pro její rozšířenost, jednoduchost a bezpečnost.

JSON

```
{
  "typ_rozhraní": "Ethernet",
  "MAC_adresa": "00:1A:2B:3C:4D:5E",
  "IP_adresa": "192.168.1.10",
  "subnet_mask": "255.255.255.0",
  "stav": "Aktivní"
}
```

XML

```
<síťové_rozhraní>
  <typ_rozhraní>Ethernet</typ_rozhraní>
  <MAC_adresa>00:1A:2B:3C:4D:5E</MAC_adresa>
  <IP_adresa>192.168.1.10</IP_adresa>
  <subnet_mask>255.255.255.0</subnet_mask>
  <stav>Aktivní</stav>
</síťové_rozhraní>
```

Obr. 34 - Ukázka formátu JSON a XML (zdroj: vlastní)

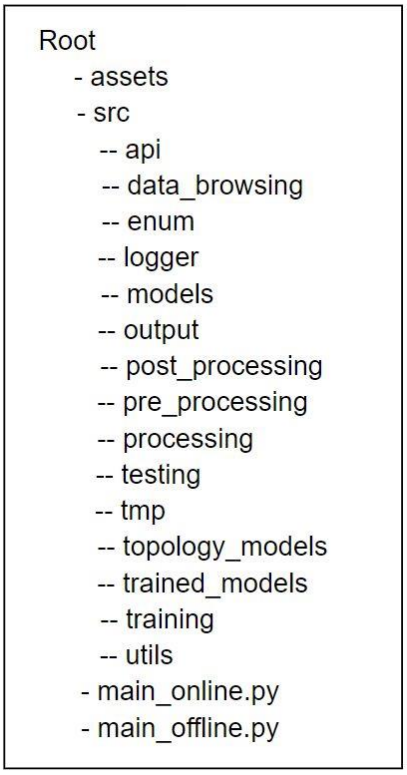
3.1 ZVOLENÝ PROGRAMOVACÍ JAZYK

Jako programovací jazyk pro tvorbu serverové části byl zvolen Python, konkrétně jeho poslední verze 3.11. Jedná se o populární, vysokoúrovňový, multiparadigmatický jazyk vytvořený v roce 1991 Guido Von Rossuem. Python je univerzální programovací jazyk, se kterým lze vytvářet širokou škálu aplikací, jako jsou webové aplikace, strojové učení, datová věda, systémová administrace, automatizace procesů, vývoj vestavěných zařízení a mnoho dalších. Python patří mezi interpretované jazyky, což znamená, že na rozdíl od kompilovaných jazyků jako je C/C++, které přeloží pomocí kompilátoru celý svůj zdrojový kód do strojového kódu, se kterým následně může operační systém pracovat, musí použít interpret. Interpret je možné představit si jako software, který čte zdrojový kód á řádek po řádku jej překládá do strojového kódu za běhu programu. Nevýhodou je potřeba nainstalovaného interpretu na cílovém počítači a pomalejší provádění kódu oproti kompilovaným jazykům, ale zato poskytuje možnost rychlejšího vývoje aplikací a lepší přenositelnost mezi různými platformami. Obecně hlavní nevýhodou interpretovaných programovacích jazyků je, že nejsou schopny odchytit chyby, které se objeví při kompilaci u jazyků jako C/C++, ale až za běhu programu. Python umožňuje využití knihoven třetích stran jako například *Numpy* (knihovna pro matematické operace) nebo *Matplotlib* (knihovna pro tvorbu grafů), které lze jednoduše instalovat pomocí systému pro správu balíčků *pip* („Preferred Installer Program“). Syntax pro instalaci je „*pip install název_balíčku*“ (Yan, 2023).

Aplikace byla vyvíjena na operačním systému Windows 10 ve vývojovém prostředí IntelliJ IDEA Ultimate od firmy Jet Brains. Projekt je zároveň verzován pomocí Gitu a uložen na GitHubu. Vývoj projektu je tak zaznamenán v podobě jednotlivých commitů a jejich zpráv.

3.2 ARCHITEKTURA SERVEROVÉ ČÁSTI

Serverová část využívá OOP („object oriented programming“) paradigma a její složková struktura odpovídá rozdělení podle funkcí („Structure By Features“). To znamená, že projekt je rozdělen na jednotlivé adresáře, kde každý z adresářů obsahuje kód rozdělený do tříd, který je nezávislý na ostatních částí projektu. Toto rozložení umožňuje lepší kohezi a sníženou míru couplingu kódu, který je tak snadněji udržovatelný a rozšiřitelný. Zároveň je tak možné jednodušeji otestovat každou část kódu, protože jsou nezávislé na ostatních. Složková strukturu znázorňuje následující obrázek.



Obr. 35 - Složková struktura
(zdroj: vlastní)

Serverová část aplikace jako taková se dělí na dvě, na sebe navzájem nezávislé aplikace. První aplikace se využívá k trénování a testování modelů sítě a uživatel s ní nemá možnost interagovat v rámci webu. Pro lepší přehlednost je tato část nazvána „Offline část aplikace“.

Druhá aplikace představuje řešení zadání práce a skládá se z modulů, se kterými uživatel může interagovat skrze webové rozhraní a je zodpovědná za zpracování vstupního obrázku a jeho vrácení v PDF podobě. Pro přehlednost je tato část nazvána „Online část aplikace“.

3.3 ONLINE ČÁST SERVEROVÉ APLIKACE

Předtím než je možné popsat princip aplikace a jak řeší jednotlivé procesy v rámci ICR, je potřeba se zaměřit na praktický problém aplikací optického rozpoznávání znaků a tou je kvalita vstupních dat. Jak bylo zmíněno v kapitole 1.3.5 , rozpoznání znaků může být v případě špatné kvality obrazu prakticky nemožné a problematika správného nastavení parametrů je velmi obsáhlé téma. Univerzální nalezení správných parametrů za každých podmínek by vyžadovalo dlouhodobější výzkum a vývoj další neuronové sítě, přímo specializované na tuto problematiku. Řešením je tak určitý kompromis, kdy vstupní data mají určitá kritéria, která musí splnit, aby bylo možné optické rozpoznání textu provést. Tato kritéria jsou následující:

- Text musí být anglický psaný tiskacím písmem (velká i malá písmena);
- Modely nepodporují detekci číslic a speciálních znaků a nerozlišují velikosti písmen (odůvodnění v kapitole 3.4.1.1);
- Text by měl být napsaný propiskou nebo centropenem, tak aby jednotlivá písmena neměla přerušované kontury;
- Text musí být psán na bílém nezmačkaném papíru bez linkování (linky je možné předkreslit tužkou pro splnění další podmínky a pak je vygumovat);
- Text musí mít konstantní výšku řádků a jednotlivých písmen;
- Písmena nesmí být namáčknuta na sebe, zároveň mezi jednotlivými písmeny nesmí být moc velká mezera (ideálně trochu menší než šířka daného písmene);
- Čárky ve větě, by měly začínat na spodní úrovni řádku, měly by být kratší než písmeno „i“ a neměly by být psány pod úhlem, ale pokud možno kolmo k řádku;
- Obrázky v rámci fotky musí být ohraničeny čtvercem/obdélníkem tlustou čárou (ideálně aspoň 2x tloušťky písma);
- Fotka by měla obsahovat pouze vyříznutý text s obrázky;
- Fotka by neměla být pootočená;
- Znaky, které nemají být detekovány (vyškrtnutý znak) musí být uzavřen do vyplněného čtverce/obdélníku;
- Fotka by měla být focena ze vzdálenosti 30-40 cm a neobsahovat stín.

3.3.1 Koncepční princip aplikace

Na webovou část aplikace je možno nahlížet jako na tzv pipelinu (rouru), která má na svém začátku vstupní data v podobě JPG / PNG formátu, na která sekvenčně aplikuje určité operace a na jejím výstupu je PDF s rozpoznaným textem. Uživatel skrze webové rozhraní nahraje obrázek na server, který jej předzpracuje a vrátí klientovi jeho originální podobu s přidaným odsazením obrázku po jeho stranách. Tím obrázek získá správné rozměry pro další operace. Na tomto obrázku pak uživatel pomocí canvasu (plátna) a přímek nastaví výšku řádků, aby bylo možné obraz efektivněji segmentovat. V rámci vývoje byla ozkoušena jak varianta Houghovy transformace, tak projekčního histogramu, ale ani v jednom případě nebyly výsledky dostatečně uspokojivé.

Obr. 36 - Ilustrace principu průchodu dat ICR pipelinou (zdroj: vlastní)

Modul předzpracování a segmentace obrazu byl dlouho testován s cílem najít optimální parametry pro obraz odpovídající kvalitě, kterou by měl mít, pakliže jsou dodrženy výše zmíněné body. Výsledkem je tak využití dvou různých binarizovaných obrázků, které obsahují vyznačené bounding boxy okolo jednotlivých písmen, jeden s odstraněným šumem a druhý bez odstraněného šumu. Pro některé fotky totiž odstranění šumu způsobuje, že dojde k poškození kontur jednotlivých písmen, což se negativně odrazí později během samotné klasifikace. V některých extrémnějších případech ani nemusí dojít k správné detekci kontur písmene, protože vlivem aplikace odstranění šumu dojde k rozpadu kontury na několik menších. Hlavní negativní vliv má zejména zmačkání papíru a stín na papíru v momentě pořizování fotografie. Uživateli jsou tak po nastavení výšky řádkování vráceny oba tyto obrázky a je na něm, aby zvolil, který obrázek má lepší kvalitu. To znamená obrázek, který má bounding boxy kolem všech písmen a zároveň, kde mají písmena neporušené kontury.

Jakmile uživatel vybere jeden z obrázků a odešle ho na server, dojde ke spuštění samotného klasifikačního procesu, kdy jsou modelu neuronové sítě postupně na vstup předkládány jednotlivé znaky nalezené během segmentace obrazu. Po klasifikaci všech znaků jsou data spojena do uceleného textového řetězce, který je naformátován (velká a malá písmena, mezery mezi jednotlivými slovy, čárky a tečky ve větách) a následně vložen na vstup modulu

post-zpracování. Po ošetření textu je výsledný text vložen do PDF souboru, kam jsou přidány obrázky (pokud se ve vstupních datech nějaký nacházel) a tento PDF soubor je vrácen klientovi, odkud si jej uživatel může stáhnout do svého počítače. Jak je vidět na obrázku č. 36 tak pipelina využívá celkem šest různých modulů, kterými data musí projít. Jednotlivé moduly jsou popsány v následujících kapitolách.

3.3.2 Modul REST API

Jak bylo zmíněno již v kapitole 3 , REST API funguje jako vstupní bod do aplikace. Pro jeho tvorbu a samotný webový server byl využit framework FastAPI. Framework je možné si představit jako skupinu knihoven, pospojovaných do takové struktury, která umožňuje vývojáři jednoduší a efektivnější vývoj aplikace. FastAPI je velmi populární framework v komunitě vývojářů webových technologií. Může za to primárně jeho rychlost, robustnost, jednoduchost, automatická dokumentace (Swagger) a využití ASGI („Asynchronous Server Gateway Interface“) architektury, která je oproti klasické WSGI („Web Server Gateway Interface“) architektuře, kterou využívají konkurenční frameworky (Flask, Django), rychlejší.

3.3.2.1 Koncové body

REST API je tvořeno tzv. koncovými body („end-point“), což jsou konkrétní URL cesty, resp. sufix na konci URL oddělený lomítkem, na které jsou směřovány požadavky klienta. Za tímto koncovým bodem se pak skrývá obslužný kód, který má vykonat požadovanou operaci a vrátit odpověď klientovi. Aplikace má celkem 4 koncové body:

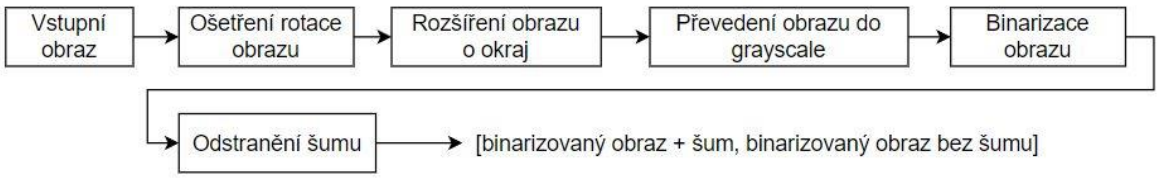
- /upload;
- /preprocess;
- /start-ocr;
- /models.

První koncový bod slouží k nahrání fotky na server a vrátí klientovi její originální variantu, ale s rozšířenými okraji, aby bylo možné na fotce nastavit výšku řádkování a její rozměry odpovídali rozměrům fotky potom, co projde předzpracováním. Druhý koncový bod slouží k provedení předzpracování a segmentaci a uživateli vrací, již zmíněné dvě varianty binarizovaného obrázku se zobrazenými bounding boxy. Třetí koncový bod slouží k provedení samotného rozpoznání znaků ve variantě obrázku, kterou uživatel vybere. Zde se kombinují jednotlivé operace rozebrané v dalších kapitolách. Server po ukončení procesu vrátí ZIP soubor, který obsahuje dva PDF soubory, první obsahuje text, jak jej rozpoznala neuronová síť

spolu s obrázky a druhý obsahuje rozpoznaný text, který prošel post zpracováním a obrázky. Poslední koncový bod vrací seznam dostupných modelů sítí, které uživatel může zvolit.

3.3.3 Modul předzpracování obrazu

Tento modul je tvořený třídou *ImgPreProcessor*, která dědí z abstraktní třídy *AbstractImgPreProcessor*, kde jsou definované jednotlivé metody, které je potřeba implementovat. Na tento modul je možné nahlížet jako na samostatnou pipelinu, kde obrazy musí projít sérií sekvenčních operací. Tyto operace jsou implementovány s využitím knihovny *OpenCV*, což je velmi populární knihovna napsaná v jazyce C++ pro manipulaci s obrazem. Její API napsané v Pythonu je dostupné v rámci balíčku *opencv-python*.



Obr. 37 - Ilustrace principu průchodu dat pipelinou pro předzpracování obrazu (zdroj: vlastní)

Jednotlivé kroky jsou implementovány v rámci metody *run_pre_processing*, která přijímá dva vstupní argumenty: systémovou cestu k uložené fotce a proměnnou typu boolean, která v metodě nastavuje, jestli se bude zprocesovávat jednotlivé písmeno nebo celková fotka. Tato proměnná je zde z důvodu, že tento modul je taktéž využíván testovacím modulem popsaným v kapitole 3.4.2.2 a v takovém případě se aplikují trochu odlišné parametry metod. Výsledky jednotlivých operací jsou pak ukládány do adresáře *tmp/img-processing/results/nazev_obrazu*. Díky tomu je možné zaznamenávat postup a výsledky jednotlivých operací, což velmi napomohlo vývoji této pipeliny.

3.3.3.1 Odstranění rotace

Prvním krokem je korekce případné nežádoucí rotace. Tuto korekci implementuje metoda *_deskew()*. Ta využívá Houghovu transformaci k nalezení přímek představující jednotlivé řádky v obrazu. Pokud není detekován úhel pootočení nebo pokud nejsou detekovány žádné přímky (například u obrazu s jediným řádkem textu), metoda vrací obraz, který dostala ve vstupním argumentu. Pokud je detekován úhel natočení tak se využije funkcí z knihovny *OpenCV getRotationMatrix2D()* a *warpAffine()*, které zajistí korekci natočení.

3.3.3.2 Rozšíření obrazu o okraj

Dalším krokem je rozšíření obrazu o okraj. Tato operace je implementována metodou *_add_padding()*. Ta využívá metodu *copyMakeBorder()* z knihovny *OpenCV* k rozšíření obrazu o 80 bílých pixelů na každé straně. Tato operace je využita z toho důvodu, že pokud je vložena fotka, která je oříznutá způsobem, že se písmena dotýkají hran obrázku, tak nemusí dojít ke správné segmentaci těchto znaků.

3.3.3.3 Převedení obrazu do grayscale a jeho binarizace

Aby bylo možné provést binarizaci (prahování) obrazu je nutné jej nejdříve převést z barevné varianty do grayscale (odstíny šedé barvy). Tuto operaci implementuje metoda knihovny *OpenCV* s názvem *cvtColor()*, která využívá vážený průměr, jak bylo zmíněno v kapitole 1.3.1 .

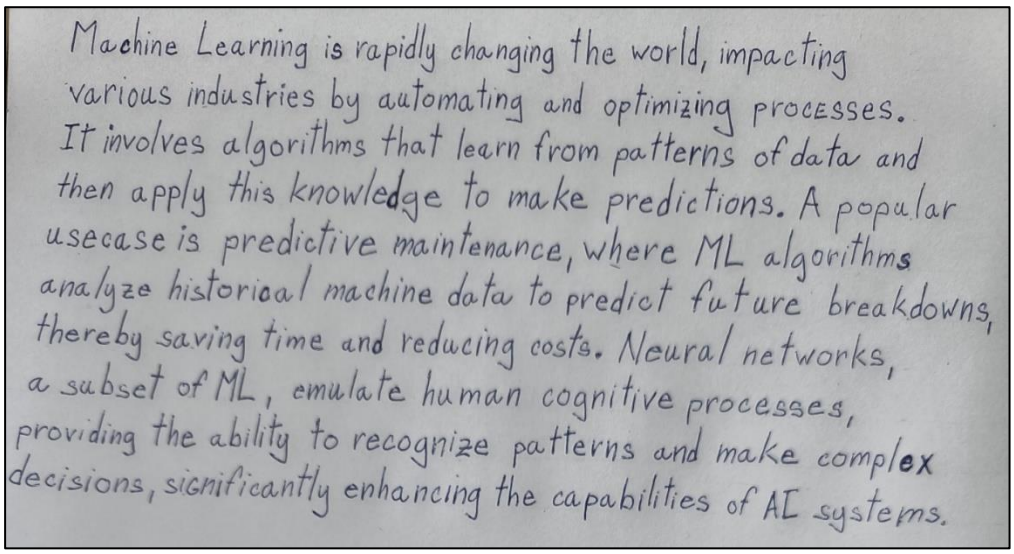
Následně dochází k adaptivní binarizaci obrazu s využitím funkce knihovny *OpenCV adaptiveThreshold(src, maxValue, adaptiveMethod, thresholdType, blockSize, C),* která je implementována v rámci metody *_apply_thresholding()*. Parametry metody mají následující význam:

- *src* – vstupní grayscale obraz, nad kterým se provede operace;
- *maxValue* – hodnota přiřazená pixelům, které překročí adaptivní práh, v tomto případě je nastavena na 255 (bílá barva);
- *adaptiveMethod* – metoda použitá k výpočtu adaptivního prahu, v tomto případě je využita metoda *ADAPTIVE_THRESH_GAUSSIAN_C*, která adaptivní práh vypočte jako vážený součet hodnot sousedství pixelu, kde váhy jsou Gaussovské;
- *thresholdType* – typ prahování, který se má použít, v tomto případě je použit typ *THRESH_BINARY* (pokud je hodnota pixelu vyšší než práh, tak je nastavena na hodnotu *maxValue*, v opačném případě je nastavena na 0);
- *blockSize* – velikost sousedství použitého k výpočtu prahu pro každý pixel, v tomto případě je tento parametr empiricky nastaven na hodnotu 65;
- *C* – konstanta odečtená od vypočítaného prahu, což přidává určitou flexibilitu v prahování, v tomto případě je hodnota empiricky nastavena na 23.

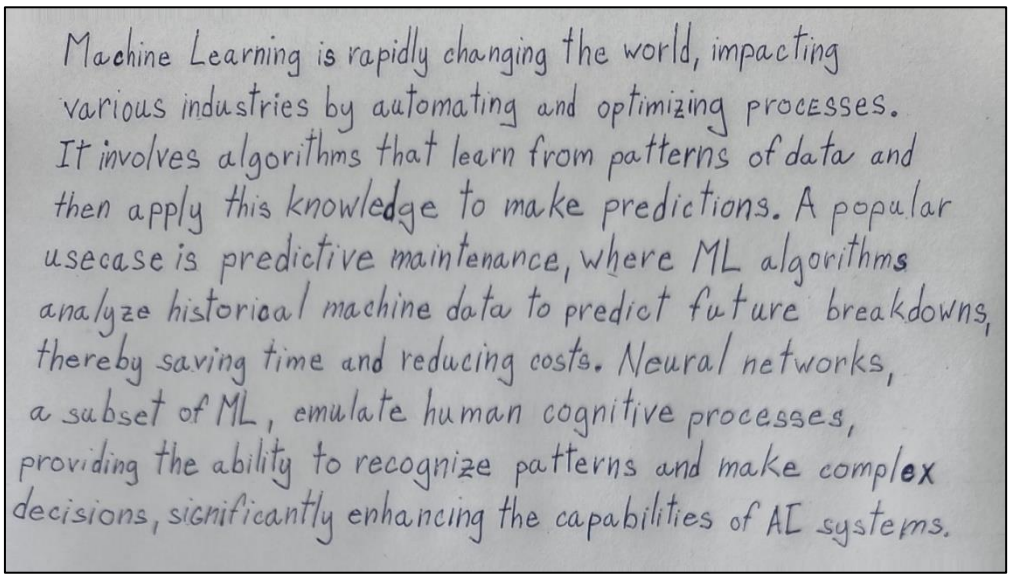
3.3.3.4 Odstranění šumu

Poslední operací je odstranění šumu implementována metodou *_remove_noise()*. Ta využívá jádra o velikosti 3x3 a sekvenčně jej aplikuje v operacích dilatace, eroze, morfologického uzavření a rozmazání pomocí mediánového filtru. Tato kombinace různých metod je opět výsledkem pokusů odstranění šumu na různých obrazech a v případě dodržení kritérií zmíněných v kapitole 3.3 funguje uspokojivě.

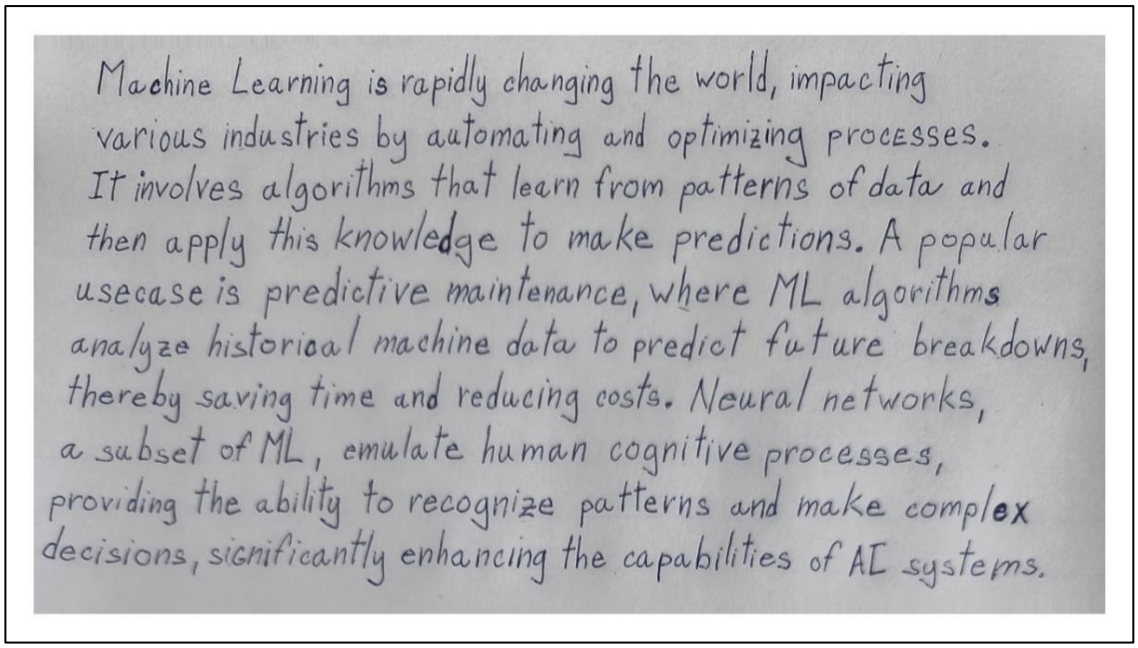
3.3.3.5 Ukázka jednotlivých výsledků průchodu pipelinou



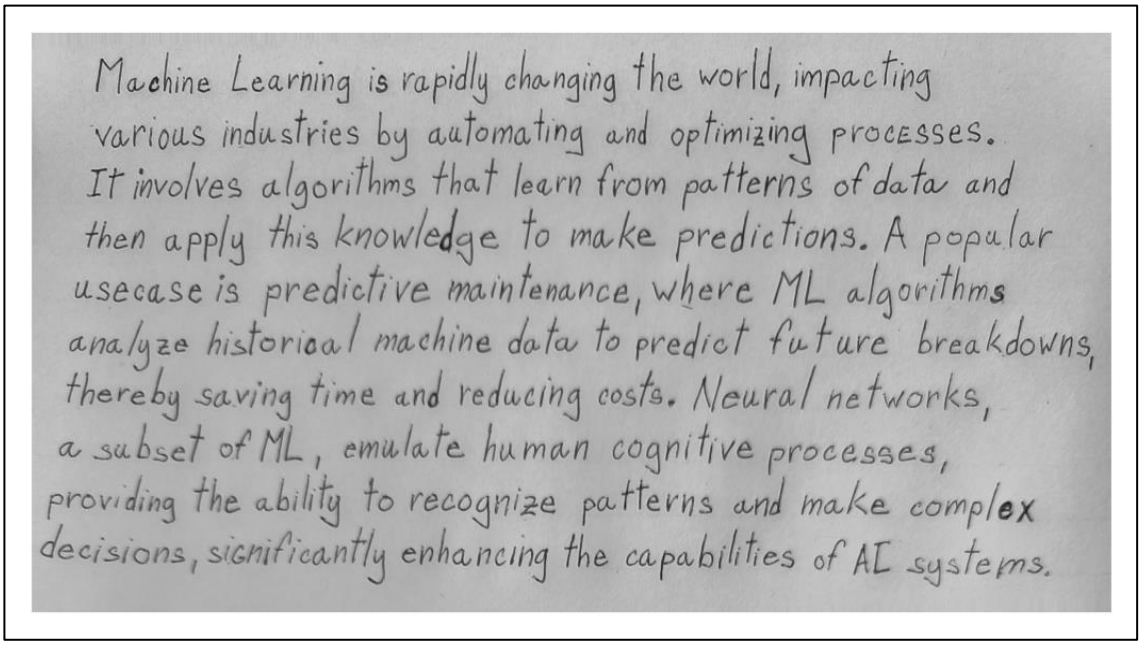
Obr. 38 - Vstupní obraz (zdroj: vlastní)



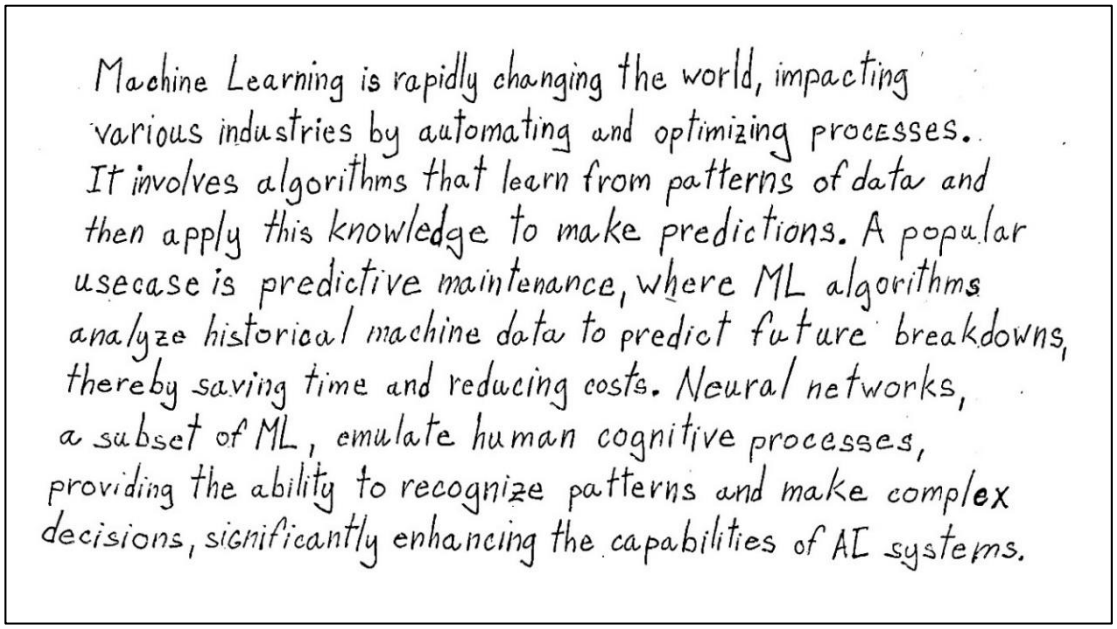
Obr. 39 - Vstupní obraz po korekci rotace (zdroj: vlastní)



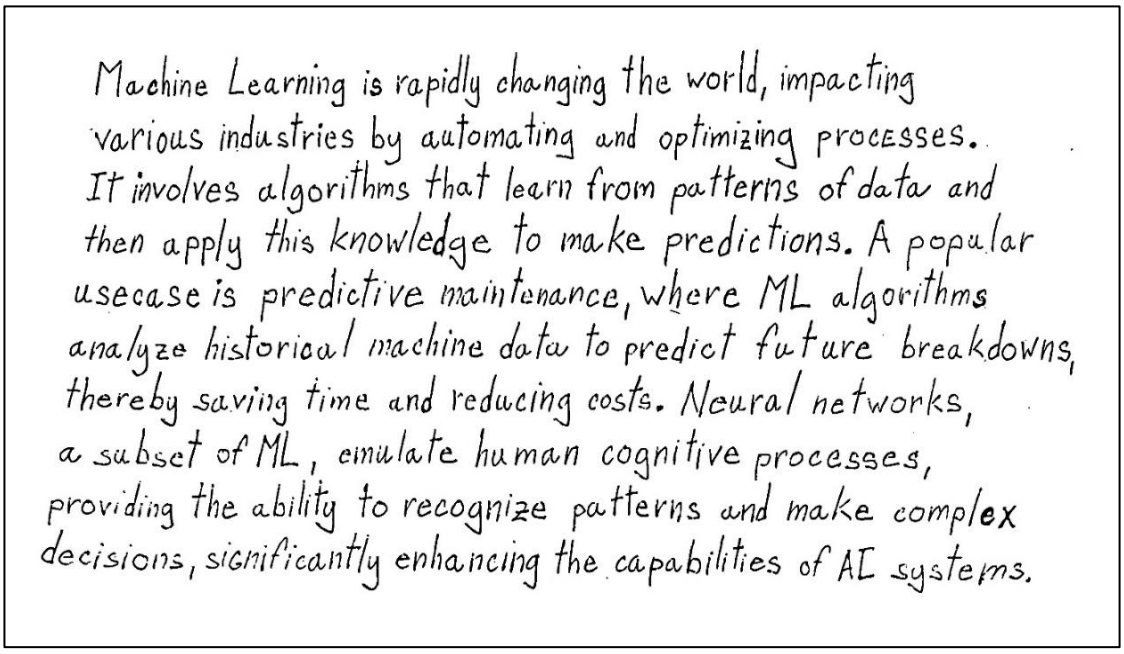
Obr. 40 - Obraz s rozšířeným bílým okrajem (zdroj: vlastní)



Obr. 41 - Obraz převedený do grayscale (zdroj: vlastní)



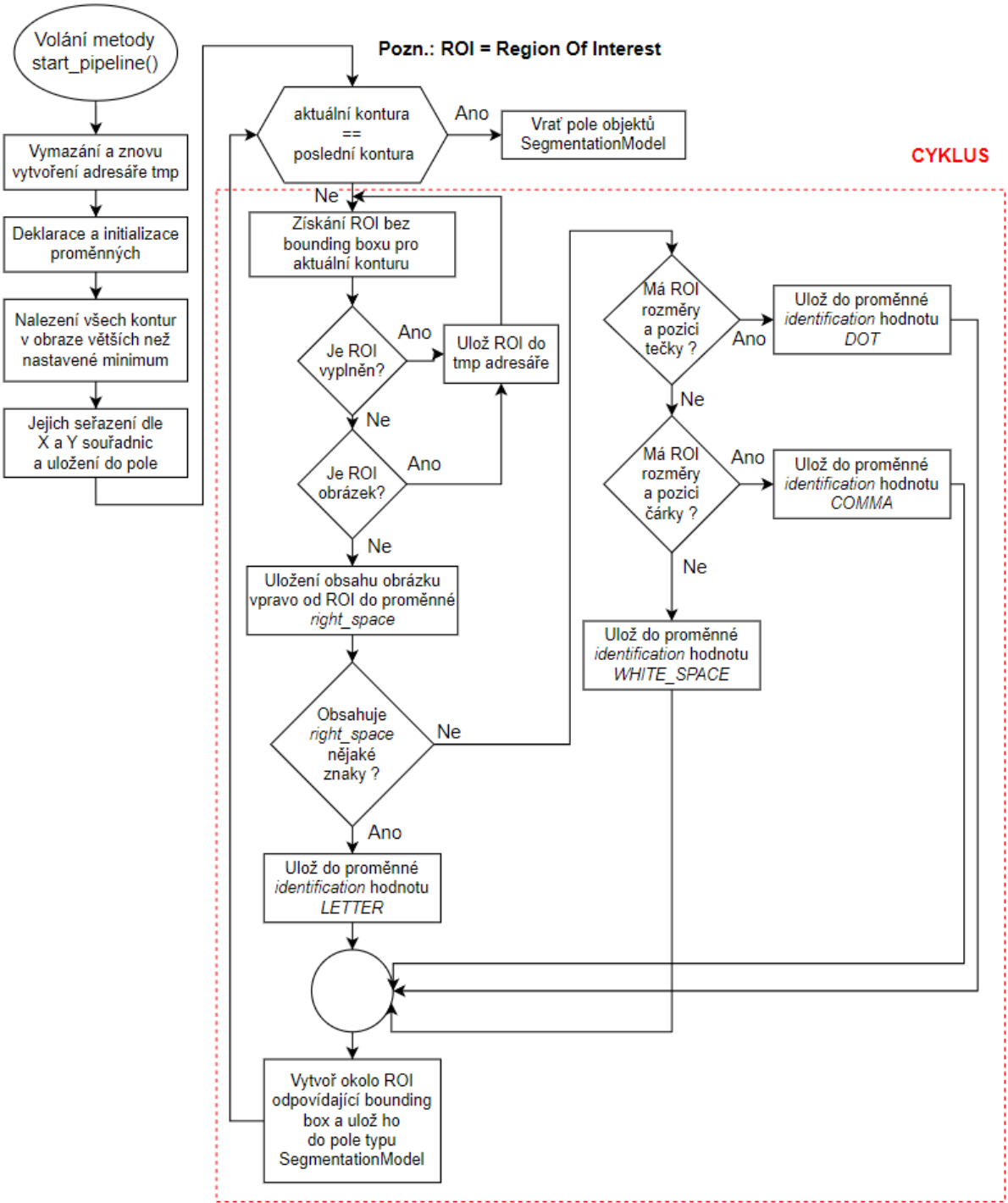
Obr. 42 - Binarizovaný obraz obsahující šum (minimální v tomto případě) (zdroj: vlastní)



Obr. 43 - Binarizovaný obraz s odstraněným šumem (zdroj: vlastní)

3.3.4 Modul segmentace obrazu

Jakmile je obraz v binarizované předzpracované formě je možné provést jeho segmentaci. O tu se stará třída *SegmentationManager* pomocí funkcionalit z knihovny *OpenCV*. Algoritmus celého procesu segmentace je zobrazen na následujícím vývojovém diagramu.



Obr. 44 - Algoritmus segmentace obrazu (zdroj: vlastní)

K nalezení všech kontur se využívá funkce *findContours()*, která implementuje Suzukiho algoritmus zmíněný v kapitole 1.4.4 . Stejně jako v případě modulu předzpracování obrazu se všechny dílčí výsledky ukládají do adresáře *tmp/img-segmentation*, kde jsou nadále roztříděny do dalších podadresářů podle typu. Jakmile jsou nalezeny všechny kontury, tak na základě hodnot, které uživatel nastavil (výška řádkování a offset v ose Y) je možné kontury seřadit s využitím X a Y souřadnic. Když jsou kontury uloženy v poli ve směru čtení, následuje další část segmentace. Ta probíhá v cyklu, kdy se z kopie obrazu (kopie, do které nezakreslují bounding boxy) vyřízne ROI („Region Of Interest“), což je obsah jednotlivých kontur.

3.3.4.1 Identifikace vyškrtnutých znaků

Nejdříve se využívá podmínka, která z textu odstraní znaky, které jsou zabarvené (přeškrtané). Z tohoto důvodu, jak bylo již dříve zmíněno, je potřeba aby takový znak byl uzavřený ve čtverci/obdélníku. Znak je kategorizován na základě poměru tmavých a bílých pixelů uvnitř ROI. Pokud je ROI víc než z 95 % vyplněný černou barvou, je znak vyřazen a nebude se rozpoznávat.

Obr. 45 - Ukázka začernění písmene pro jeho vyřazení z klasifikace (zdroj: vlastní)

3.3.4.2. Identifikace obrázků

V dalším kroku se podmínkou zjišťuje, jestli aktuální ROI není obrázek. Tato kontrola funguje jednoduše na principu porovnání velikostí ROI aktuálního vzorku a průměrné velikosti vzorku. Pokud je aktuální ROI větší než 10ti násobek průměrné hodnoty, je tato oblast identifikována jako obrázek. Ten je na základě souřadnic vyříznut z původního barevného obrázku uloženého v adresáři *tmp* (ne jeho binarizované podoby) a uložen zpět do adresáře

tmp/img-segmentation/pictures. Díky tomu je možné ho na konci pipeliny přidat do PDF souboru. V případě, že je ROI identifikován jako začerněný znak nebo jako obrázek, dojde k ukončení současné iterace cyklu a cyklus pokračuje další iterací.

3.3.4.3 Identifikace zbytku prohledávaných oblastí

V tento moment se algoritmus dostává ke své stěžejní části a tou je identifikace ROI na základě rozměrů a umístění v rámci řádku. Znaky se ve výsledku ukládají do pole tvořeného objekty typu *SegmentationModel*. Každý uložený segmentovaný znak má tak spolu se svojí bitmapou spojený identifikátor určený pomocí enumerační třídy *IdentificationEnum*. Ta deklaruje, že jednotlivé znaky mohou být buď tečky na konci věty, čárky ve větě, písmena nebo písmeno po němž následuje mezera.

```
@dataclass
class SegmentationModel:
    bitmap: numpy.ndarray
    identification: IdentificationEnum


class IdentificationEnum(IntEnum):

    DOT = 0
    COMMA = 1
    LETTER = 2
    WHITESPACE_LETTER = 3
```

Obr. 46 - Třída SegmentationModel (zdroj: vlastní)

Aby bylo možné klasifikovat dle tohoto rozdělení, je vytvořena proměnná *right_space*, což je bitmapa, která představuje oblast vpravo od ROI se stejnými rozměry. Pokud je poměr bílých pixelů vyšší než 98%, tak se aktuální ROI může klasifikovat buď jako tečka, čárka nebo písmeno po němž následuje mezera. V případě, že je poměr bílých pixelů nižší než 98 %, tak je uvažováno, že se vpravo od prohledávané oblasti nachází další znak, a tudíž je aktuální ROI identifikován jako písmeno nastavením proměnné *identification* na hodnotu *LETTER*.

Nejdříve se podmínkou testuje, jestli má aktuální ROI rozměry odpovídající tečce a jestli se nachází v rámci řádku na jeho spodní hranici (bráno dle Y hodnoty). Pokud je podmínka vyhodnocena jako *False*, vstupuje cyklus do další podmínky, která testuje, jestli ROI

odpovídá rozměry a Y souřadnici čárce. Pokud je i tato podmínka vyhodnocena jako *False,* je znak identifikován jako písmeno, po němž následuje mezera.

Rozměry čárek a teček ve větě jsou definované s využitím vzorového obrázku, na kterém byly nastaveny takové parametry segmentace, aby pro dané rozměry teček a čárek byl algoritmus schopný tyto znaky spolehlivě segmentovat. Pro příklad v tomto vzorovém obrázku byla průměrná výška a šířka tečky okolo 5 px a její ROI tak lze uvažovat jako čtverec. Aby se započítal fakt, že každý člověk píše tečky ve větě jinak tlusté, došlo k rozšíření hodnot, které mohly být považovány za tečku od 4 px do 7 px. Cokoliv většího už v testovaném obraze nevypadalo jako tečka na konci věty. Tečka taky nemusí mít nutně čtvercový tvar ROI, a tak je jako tečka zaznamenána jakákoliv oblast ROI, jejíž výška a šířka spadala do rozsahu 4 – 7 px a všech možných permutací těchto hodnot (tzn. 4x4, 4x5, 4x6, 4x7, 5x4 atd.). Stejným způsobem je postupováno u čárek, u kterých se předpokládá, že jejich ROI bude mít tvar obdélníku, kde rozsah jeho hodnot pro výšku je 9 px až 16 px a rozsah hodnot šířky je 1 px až 6 px.

3.3.4.4 Ošetření problematiky různě velkých vstupních dat

Tyto konkrétní hodnoty ale mají význam jen pro tento konkrétní vzorový obraz, protože obrazy s jinými rozměry, budou mít také jiné rozměry písma, čárek a teček. Protože vstupní fotka má být ve formě pouze oříznutého textu (a obrázků), tak je možné svázat tyto konkrétní hodnoty s velikostí obrázku. To znamená, že je možné vytvořit konstanty, kterými lze vynásobit šířku a potažmo výšku fotografie, a dosáhnout tak získání rozsahu rozměrů, které pro daný obraz odpovídají tečkám a čárkám ve větě. Stejným způsobem je řešena minimální hodnota rozeznávaných kontur (hodnota, kdy se daná kontura registruje a kdy už se bere jako zbytkový šum a ignoruje se). Ve vzorovém obrazu tato hodnota odpovídala 4 px, a protože původní obraz měl velikost 765 578 px, tak pro výslednou násobící konstantu platí vztah:

$$\_CONTOUR\_SCALE\_CONSTANT = \frac{4}{765\,578} = 5.2 \cdot 10^{-6} \tag{17}$$

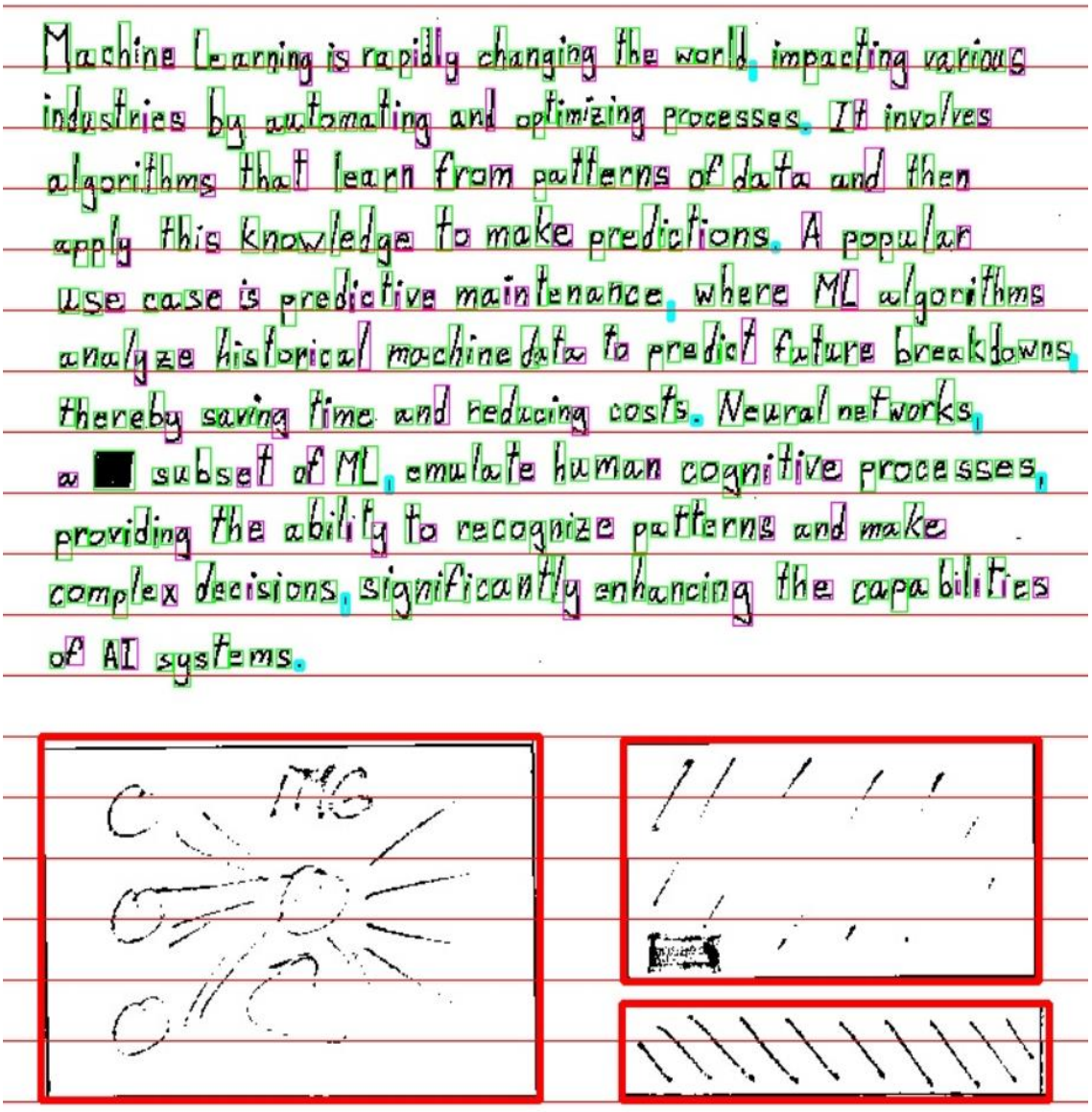
Když se touto hodnotou vynásobí velikost vstupní fotky (výška · šířka), tak jejím výsledkem je minimální plocha kontury.

Stejného přístupu se využívá v nastavení šířky mezery mezi jednotlivými slovy, protože některá písmena jako „i“ nebo „l“ mají menší hodnotu proměnné *right_space* a tímto způsobem se tento fakt kompenzuje. Když by totiž autor klasifikovaného textu udělal moc velkou mezeru mezi jedním z těchto písmen a dalším písmenem, mohlo by dojít k situaci, kdy písmeno bude identifikováno jako písmeno na konci slova a při rekonstrukci textu by tak na místě slova došlo

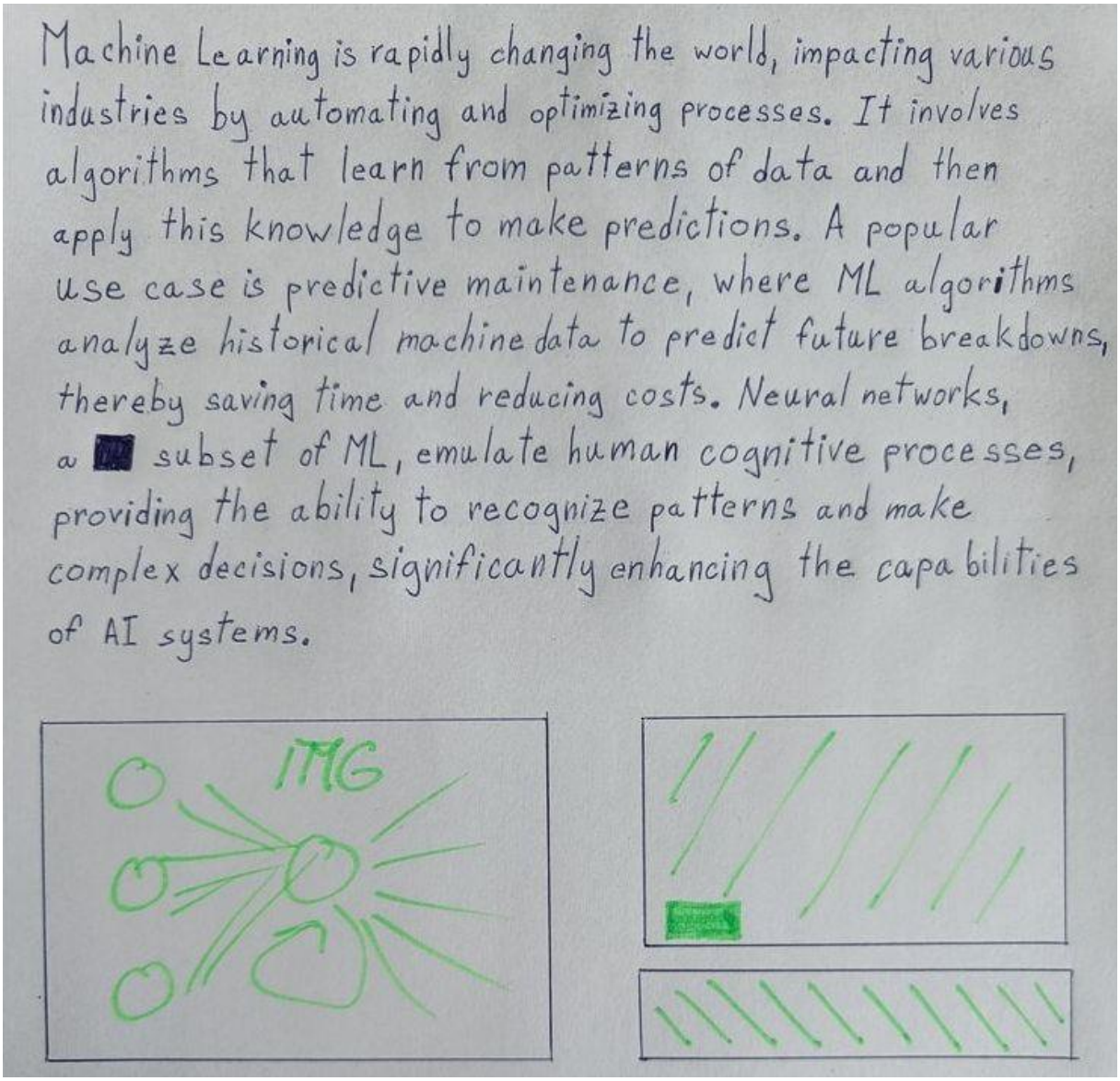
k rozdělení mezerou. Z toho důvodu mají písmena, jejichž ROI má stejnou šířku jako čárka ve větě nastavenou větší šířku prohledávaného prostoru v proměnné *right_space*.

3.3.4.5 Ukázka výsledku segmentace

Jakmile algoritmus identifikuje aktuálně zpracovávaný ROI, nakreslí kolem něj bounding box. Zelenou barvou jsou označena normální písmena, fialovou písmena na konci slova, tyrkysovou čárky a tečky na konci věty a červenou barvou obrázky.



Obr. 47 - Ukázka segmentace obrazu s vyznačeným řádkováním (zdroj: vlastní)



Obr. 48 - Vstupní obraz, na který se aplikovaly operace předzpracování a segmentace obrazu (zdroj: vlastní)

3.3.5 Klasifikační modul

Po dokončení segmentace obrazu jsou data ještě dodatečně upravena přidáním okrajů o velikosti 4px po každé straně a následně je vzorek upraven do rozměrů 28x28 pixelů. Vzorek se ještě převede z podoby, kdy je černý text na bílém pozadí na bílý text na černém pozadí, což je formát, ve kterém jsou uložena data v EMNIST datasetu. Následně je pole těchto vzorků předáno modulu pro klasifikaci jednotlivých znaků.

V tomto modulu, který je tvořen třídou *PredictionManager* se deklaruje nové pole *_results.* Předané pole se poté prochází v cyklu a objekty s identifikátorem označující tečku a čárku se tak mohou do pole *_results* uložit rovnou bez nutnosti využití neuronové sítě. Bitmapy, jejichž identifikátor je buď *LETTER* nebo *WHITESPACE_LETTER*, jsou pak předány na vstup zvoleného modelu neuronové sítě ke klasifikaci. V případě klasifikace bitmapy spojené s identifikátorem *WHITESPACE_LETTER* se po uložení rozpoznané hodnoty do pole *_results* následně ještě uloží znak mezery. Toto pole následně projde dalším cyklem, který naformátuje velikosti písmen a mezer mezi znaky tak, aby odpovídaly standartní stavbě věty. To znamená velká písmena na začátku věty, mezery po čárkách ve větě a ukončení věty tečkou, za kterou následuje mezera. Aplikace tedy neumožňuje rozeznávání jmen a názvů, takže jsou zapsány malým písmem.

3.3.5.1 Potenciální řešení rozeznání velkých písmen názvů

Problém rozpoznávání velkých písmen v názvech by mohl být řešen přidáním další metody do modulu post-zpracování obrazu, která by obsahovala slovník slov, ve kterém by byla uložena jména, potažmo názvy míst. Tím by bylo možné ošetřit fakt, že síť není schopna rozlišit mezi velkým a malým písmem, jak bude zdůvodněno v kapitole 3.4.1.1.

Výsledkem tohoto modulu je ucelený textový řetězec, který už formátováním odpovídá psanému lidskému textu, tak jak jednotlivé znaky klasifikoval model neuronové sítě.

3.3.6 Modul post-zpracování textu

Tento modul reprezentovaný třídou *OpenaiOCRPostprocessing* je posledním kusem kódu, kde dochází k manipulaci s daty. Využívá dvě různé varianty úpravy textu. Tou první je lexikální korekce textu s využitím knihovny *pyspellchecker*. Textový řetězec získaný z klasifikačního modulu je rozdělen na jednotlivá slova pomocí *split()* funkce. Tato slova jsou následně porovnána se slovníkem existujících slov. Funkce *correction(word)* třídy

SpellChecker() v argumentu přijímá slovo a vrací slovo, které vyhodnotí jako nejpravděpodobnější variantu. V případě, že je jako argument předáno správně zapsané slovo, funkce neprovede žádnou operaci a uloží do pole opravených slov původní slovo. Výsledné pole opravených slov se pak následně opět spojí do naformátovaného textového řetězce.

3.3.6.1 Využití generativního modelu pro korekci výsledku ICR

Druhou variantou je využití API velkého jazykového modelu (LLM) od firmy OpenAI s názvem GPT3.5–Turbo. Jedná se o zpoplatněnou službu, kde si uživatel může vybrat z množství vytrénovaných generativních modelů firmy OpenAI. Platí se za využití počtu tokenů, které představují základní jednotku textu, kterou model zpracovává. Token tak může být jedno písmeno nebo i slovo, a to jak na vstupu (text co zašlete modelu) tak na výstupu (délka textu odpovědí generovaná modelem). Cena a maximální počet tokenů záleží na jednotlivých modelech. Pro využitý model GPT3.5–Turbo je ceník následující:

Model	Input	Output
4K context	$0.0015 / 1K tokens	$0.002 / 1K tokens
16K context	$0.003 / 1K tokens	$0.004 / 1K tokens

Obr. 49 - Ceník využití API modelu GPT3.5–Turbo (https://openai.com/pricing, 2023)

Princip této varianty korekce je takový, že na vstup modelu se skrze veřejného API v definovaném formátu odešle prompt (vstupní dotaz/požadavek) s následujícím textem.

„*Following sentences are result of OCR and contain lexical errors and malformations. Please refactor it so the context of the sentences makes sense. Put $_START_$ sequence before start and $_END_$ sequence after the end of whole text, so it can be parsed. Do not format text any other way, return the result as plain text. Here are the sentences:* “

Za tímto vstupním požadavkem se vloží opravený text získaný během první varianty korekce. Model následně vygeneruje patřičnou odpověď s rekonstruovaným textem ohraničený požadovanou sekvencí *$_START_$* a *$_END_$*. Tato odpověď je následně parsována a uložena do textového souboru v adresáři *output*.

Následující tři podkapitoly porovnávají výsledky obou korekcí vůči naformátovanému textu na výstupu z neuronové sítě.

3.3.6.2 Naformátovaný výsledek z neuronové sítě

„Ma*chne learnng is rapi df y changihg the worid, impafti ng varims industri es byautomati ng ahd optimzing processes. It involves aigorifhms that tearn from patterns ofdata and fhen appjy ths knowyedge to make predicti ons. A popuiar use case g predi c ti ve mai n tenanfe, where ml aigorifhms anahy ze hisfort cal machinedata to predicf futuxe breaildowns, thereby saving ti meand reducing costs. Neurainetworks, a dsubse t of ml, emuiate human cognf ti ve processes, providing the abtii ty to recoghrze patterns and make compiex deci sjons, significaatyy enhahcifjg fhe capabl ii ti es of ai systems.“*

3.3.6.3 Text po lexikální korekci

„machine learning is rape of y changing the worid, impact no varies industry es byautomati no and optimizing processes. It involves algorithms that learn from patterns data and when apply the knowledge to make predict ons. A popular use case i pedi i ti ve mai i tenanfe, where my algorithms any ze history cal machinedata to predict future breaildowns, thereby saving ti meand reducing costs. Neurainetworks, a double i of ml, emulate human conf ti ve processes, providing the aii ty to recognize patterns and make complex deck sjons, significantly enhahcifjg fhe capable ii ti es of ai systems.“

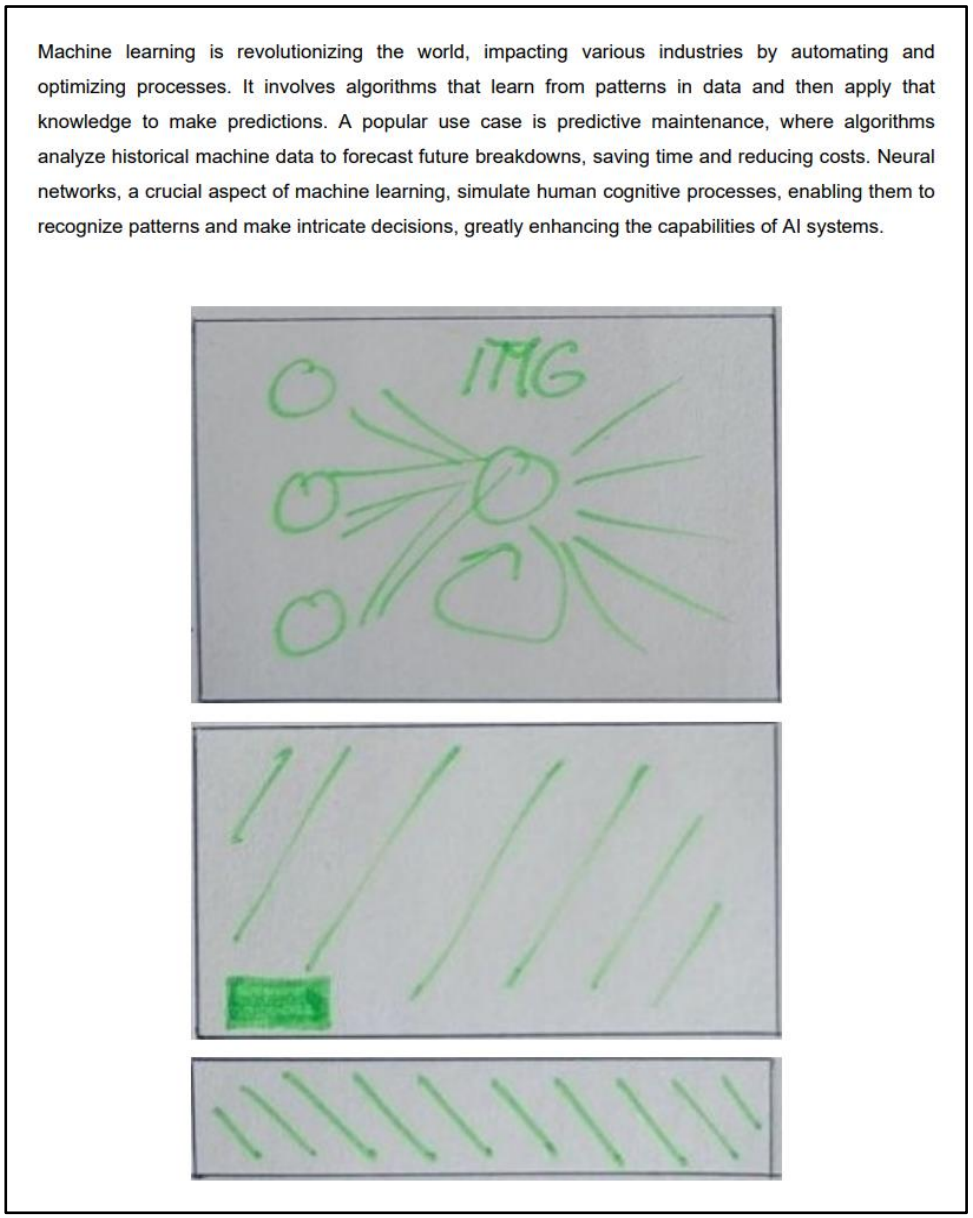
3.3.6.3 Zrekonstruovaný text model GPT3.5–TURBO

„Machine learning is revolutionizing the world, impacting various industries by automating and optimizing processes. It involves algorithms that learn from patterns in data and then apply that knowledge to make predictions. A popular use case is predictive maintenance, where algorithms analyze historical machine data to forecast future breakdowns, saving time and reducing costs. Neural networks, a crucial aspect of machine learning, simulate human cognitive processes, enabling them to recognize patterns and make intricate decisions, greatly enhancing the capabilities of AI systems.“

Na zrekonstruovaném textu je vidět stochastická povaha generativních modelů. To znamená, že pokud model zpracovává zdeformované slovo a vloží ho do kontextu celé věty, pak se může rozhodnout použít úplně jiné slovo, než kterému je zpracovávané slovo podobné.

3.3.7 Modul transformace textu do PDF

Tento modul je finální částí ICR pipeliny, která formátuje data uložená v adresáři *output* do PDF souboru. K tomu je využito knihovny *reportlab*, která umožňuje jednoduchou konverzi dat v .TXT do PDF souboru. Data jsou zarovnaná do bloku s odsazením od okrajů po obou stranách. Po vložení textu se na konec PDF souboru vloží obrázky, které byly vyříznuty z obrazu v segmentačním modulu zmíněným v kapitole 3.3.4.2. Obrázky jsou tak načteny z adresáře *tmp/img-segmentation/pictures* a pokud je potřeba, je před vložením do PDF upravena jejich velikost se zachováním poměru stran. Ve výsledku jsou vytvořeny dva PDF soubory s názvem *postprocessed_data.pdf* a *raw_data.pdf* a jsou vráceny klientovi v podobě jediného ZIP souboru.



Obr. 50 - Výsledný PDF soubor (zdroj: vlastní)

3.4 OFFLINE ČÁST SERVEROVÉ APLIKACE

3.4.1 Modul trénování

Tento modul je součástí první aplikace a spadá sem obsah složek: *training*, *topology_models* a *trained_models*. Tato část umožňuje trénink jednotlivých topologií sítí prostřednictvím knihovny Tensorflow (viz kapitola 3.4.3). Předtím, než je možné vytrénovat model s danou topologií, je nutné topologii vytvořit v samostatném souboru a přidat ji do adresáře *topology_models*. Jednotlivé soubory v tomto adresáři obsahují funkce, které vrací topologii modelu při zavolání. Proces trénování je spolu s procesem testování možné začít spuštěním souboru *main_offline.py*. Uživatel je po spuštění tohoto souboru vyzván prostřednictvím konzole, aby zvolil, jestli chce vytrénovat nový model sítě nebo otestovat jeden ze stávajících modelů uložených ve složce *trained_models*.

3.4.1.1 Využitý dataset

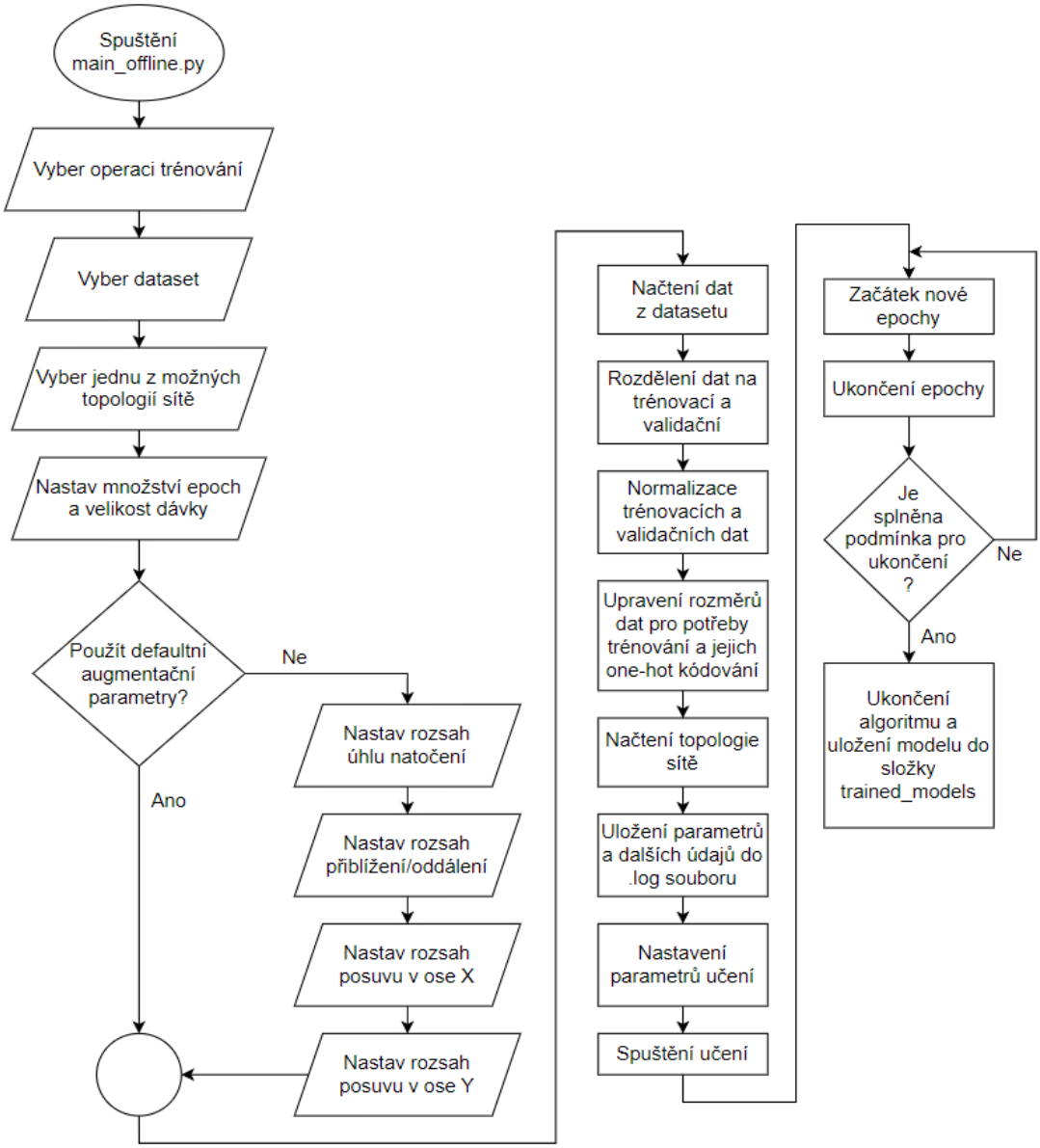
Pro trénování bylo využito datasetu EMNIST, což je rozšíření známého datasetu „NIST Special Database 19“. EMNIST obsahuje obrázky ručně psaných velkých a malých latinských písmen a číslic o rozměrech 28x28 pixelů a je dostupný buď jako CSV soubor nebo jako balíček v rámci Pythonu. Více informací o tomto datasetu lze získat v publikaci „EMNIST: an extension of MNIST to handwritten letters“ (Cohen, 2017).
Tento set dat má 6 následujících rozdělení:

- EMNSIT ByClass: 814 255 znaků, 62 nevyvážených tříd;
- EMNIST ByMerge: 814 255 znaků, 47 nevyvážených tříd;
- EMNSIT Balanced: 131 600 znaků, 47 vyvážených tříd;
- EMNIST Letters: 145 600 znaků, 26 vyvážených tříd;
- EMNIST Digits: 280 000 znaků, 10 vyvážených tříd;
- EMNIST MNIST: 70 000 znaků, 10 vyvážených tříd.

Aplikace využívá variantu *Letters* z důvodu vyváženosti třídy a není tak schopna detekovat číslice, ale pouze písmena. V prvotní fázi vývoje aplikace byla využita varianta *ByClass* obsahující znaky velkých a malých písmen a číslic. Jelikož tato varianta není vyvážená, to znamená, že obsahuje mnohem větší množství číslic než ostatních znaků, tak docházelo k přeučení v rámci vzorků číslic. To vedlo k tomu, že síť ve větší míře nesprávně klasifikovala jednotlivé znaky jako číslice. V rámci vývoje došlo k ošetřením typu penalizace tříd, které byly v datasetu zastoupeny ve značně větším množství než ostatní vzorky, ale toto opatření nevedlo

ke kýženému výsledku. Po sérii podobných ošetření tak bylo rozhodnuto použít variantu *Letters*, která vykazovala mnohem lepší výsledky za cenu, že síť není trénovaná na rozpoznávání číslic a rozlišování mezi malým a velkým písmem. Za úpravu velikosti písma je tak zodpovědný dříve zmíněný klasifikační modul. Zajímavým faktem je, že snížení počtu tříd z 62 na 26, nemělo negativní vliv na kvalitu rozeznání velkých a malých písmen v rámci stejné třídy. Obecný předpoklad byl takový, že pokud bude mít síť možnost naučit se zvlášť tvar malého a velkého písmena pro danou třídu, tak dojde ke zlepšení kvalitě klasifikace. Nicméně k potvrzení toho předpokladu nedošlo, minimálně ne v rámci testování modelů, které byly trénovány v rámci této práce.

V projektu je tak možné ve složce *training* najít dva soubory, kde první soubor s názvem: *training_manager_for_emnist_csv_dataset.py* byl použit pro trénování na datasetu *ByClass* a druhý soubor s názvem *training_manager_for_emnist_package.py*, který byl použit pro trénování na datasetu ve variantě *Letters*. Na následujícím obrázku je zobrazen vývojový diagram, popisující trénovací cyklus.



Obr. 51 - Vývojový diagram trénovacího algoritmu

3.4.1.2 Augmentace vstupních dat

Jak ukazuje vývojový diagram na obrázku č. 36, poté, co uživatel vybere dataset a topologii sítě, tak má možnost nastavit augmentační parametry. Tyto parametry mají za úkol upravit vstupní obraz – náhodným způsobem ho otáčet o daný úhel, přibližovat nebo oddalovat

a posouvat po ose X a Y. Účelem těchto operací je předejít přeučení sítě a zároveň zvýšení její robustnosti. V rámci vývoje bylo otestováno množství různých kombinací parametrů, kde výsledné parametry nakonec pro většinu nově trénovaných modelů zůstaly na výchozích hodnotách, které byly nastaveny následovně:

- Rozsah natočení: ±15 °;
- Rozsah přiblížení / oddálení: ±15 %;
- Posuv ve směru osy X: 0.1;
- Posuv ve směru osy Y: 0.1.

Pro některé modely bylo dosaženo lepší výsledné přesnosti pomocí nastavení větších hodnot augmentačních parametrů. Problémem je, že hledání vhodného modelu neuronové sítě je dlouhý dynamický proces, který vyžaduje velké množství času a vysoký počet vytrénovaných modelů jednotlivých topologií s různými parametry a topologickými úpravami. Z tohoto důvodu, aby bylo možné účinnost modelů efektivně porovnat nebyly augmentační parametry pro jednotlivé modely měněny.

3.4.1.3 Úprava dat a jejich rozdělení

Po načtení dat z datasetu dochází k jejich rozdělení na trénovací a validační množinu v poměru 80:20. Výhodou EMNIST datasetu je, že v rámci emnist knihovny existuje metoda, která umožňuje extrahovat zvlášť subset určený k trénování a subset určený k testování. Na trénovací a validační množinu tak lze rozdělit pouze první subset a stále mít izolovanou sadu dat, kterou neuronová síť neviděla a může se díky tomu využít k otestování sítě. Následně se provede normalizace dat a jejich one-hot kódování. Po načtení topologie sítě dochází k nastavení parametrů učení, což zahrnuje následující úpravy. Nastavení politiky smíšené přesnosti (při využití GPU) na *mixed_float_16*, ta kombinuje *float16* a *float32* operace a je obecně rychlejší a méně paměťově náročná než běžný *float32*. Následně se nastavuje optimalizační algoritmus učení.

3.4.1.4 Nastavení parametrů učení

V rámci vývoje opět došlo k porovnání různých typů optimalizačních algoritmů s různě nastaveným parametrem *learning_rate*. Optimalizační algoritmy byly testovány v rámci dvou různých topologií sítí (topologie budou rozebrány dále) a jsou uvedeny v následujícím seznamu.

- RMSprop – učení vykazovalo vysokou fluktuaci mezi jednotlivými epochami u hodnot parametrů *val_loss* a *val_accuracy;*
- SGD – učení stále vykazovalo fluktuaci hodnot zmíněných parametrů, ale v nižší míře než v případě RMSprop algoritmu;
- Adagrad – vykazoval z testovaných variant nejlepší výsledky s minimální fluktuací hodnot a nejvyšší dosaženou mírou přesnosti;
- Adadelta – podobné výsledky jako SGD algoritmus;
- Adamax – stejně jako RMSprop vykazuje vysokou fluktuaci hodnot parametrů;
- Adam – defaultní algoritmus s nejvíce konstantními výsledky.

Tyto závěry byly dosaženy vzhledem ke konkrétním topologiím sítě a nastavení parametrů procesu učení. Nalezení nejvhodnějšího optimalizačního algoritmu je složitý proces, který by mohl obsáhnout samostatnou závěrečnou práci. Z tohoto důvodu a vzhledem k výsledkům byl nakonec pro učení výsledných modelů použit algoritmus Adam, díky jeho obecně nejlepším výsledkům pro různé další typy topologií. Výsledné nastavení je tak zobrazeno na následujícím obrázku.

```
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
self._data_augmentation_generator.fit(self._train_X)
early_stop = EarlyStopping(monitor='val_loss', patience=13, restore_best_weights=True)
lr_reduction = ReduceLROnPlateau(monitor='val_loss',
                                 patience=6,
                                 verbose=1,
                                 factor=0.5,
                                 min_lr=1e-6)
```

Obr. 52 - Nastavení parametrů modelu před začátkem trénování (zdroj: vlastní)

Optimalizační algoritmus Adam zde využívá kategorickou křížovou entropii jako chybovou funkci. Pro definování výkonosti modelu se využívá metrika přesnosti. Parametr *learning_rate* je v počáteční fázi nastaven na hodnotu 0.001. Funkce *ReduceLROnPlateau* pak zajišťuje, že pokud nedojde ke snížení hodnoty parametru *val_loss* během šesti následujících epoch, tak dojde ke snížení hodnoty parametru *learning_rate* na polovinu, tedy 0.0005. Pokud hodnota parametru *learning_rate* klesne pod hodnotu definovanou proměnnou *min_lr* (10^{-6}),

dojde k ukončení trénování. Snížení rychlosti, s jakou se model učí se využívá ke snížení šance, že dojde k přeučení modelu. V případě, že nepomůže snížení rychlosti učení, je implementována metoda *EarlyStopping*. Ta v případě, že nedošlo ke snížení hodnoty parametru *val_loss* v 13ti po sobě jdoucích epochách, ukončí trénování a uloží model s váhami, které odpovídají nejnižší dosažené hodnotě parametru *val_loss*. Hodnota 13 je nastavena proto, aby měl model možnost dvakrát provést snížení parametru *learning_rate*, než dojde k zastavení procesu.

3.4.2 Modul testování

Tento modul je rovněž součástí první aplikace a slouží k otestování přesnosti vytrénovaného modelu. Využívá k tomu dva různé přístupy. Adresář *testing* obsahuje tři soubory: první soubor, který byl použit k testování na datasetu ve variantě *ByClass*, druhý soubor s názvem *testing_manager_for_emnist_package.py* a třetí soubor *testing_manager-_external_images.py*.

3.4.2.1 Testování pomocí testovacího datasetu EMNIST

Soubor *testing_manager_for_emnist package.py* využívá separaci tréninkových dat, jak bylo zmíněno v kapitole 3.1.2.3. Po spuštění tohoto souboru dojde k načtení dat a jejich normalizaci. Samotné testování zajišťuje knihovna *sklearn.metrics* pomocí funkce *classification_report*, která se využívá pro zobrazení hlavních klasifikačních metrik pro binární nebo multiclass klasifikační problémy. Funkce přijímá jako parametry skutečné a předpovězené hodnoty a vrací textový řetězec, který obsahuje hodnoty metrik precision, recall, f1-score a support pro každou třídu, stejně jako průměrné hodnoty těchto metrik. Tento soubor lze spustit samostatně (ne prostřednictvím souboru *main_offline.py*).

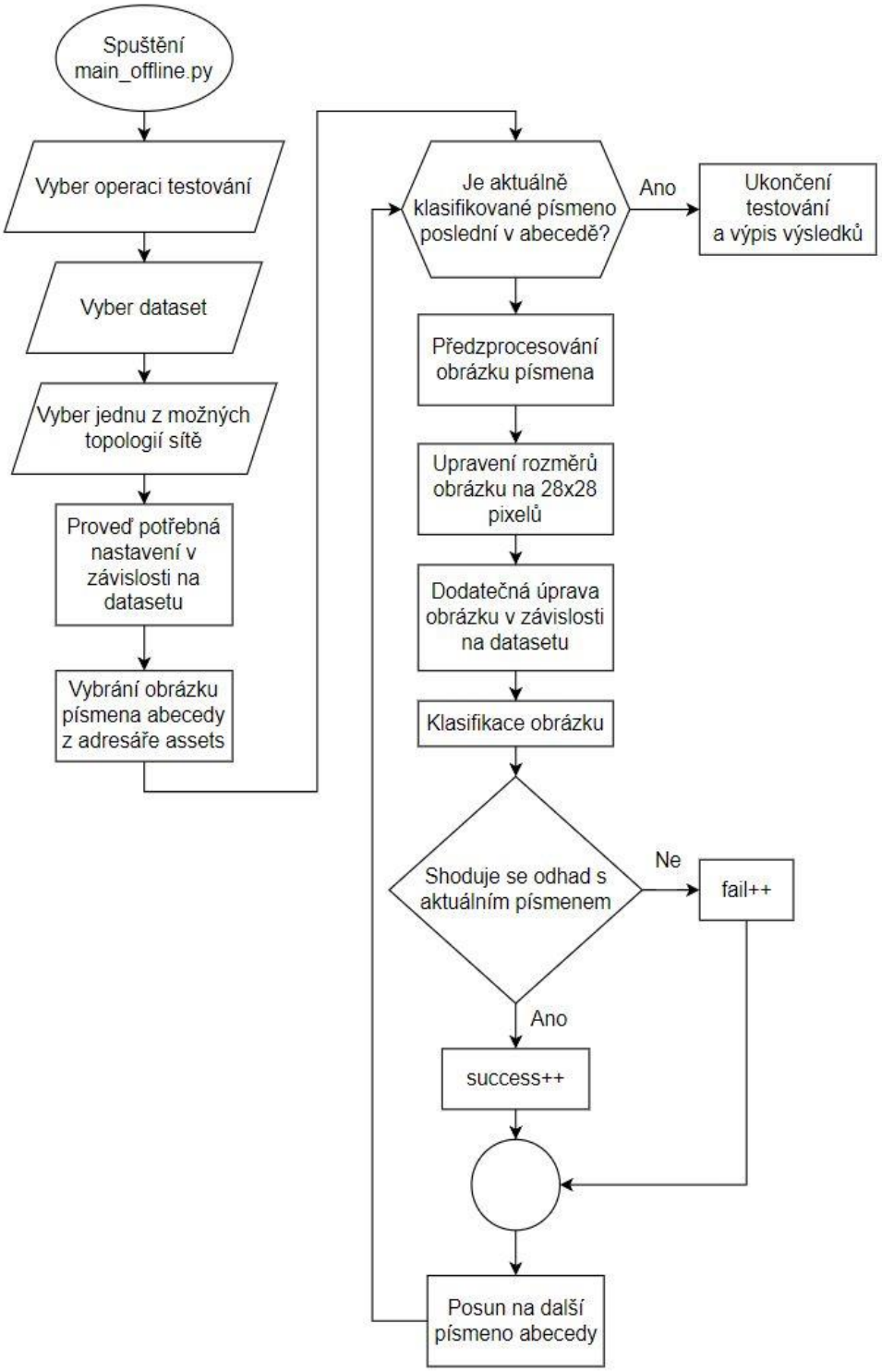
	precision	recall	f1-score	support
0	0.94	0.96	0.95	800
1	0.98	0.97	0.97	800
2	0.98	0.96	0.97	800
3	0.95	0.98	0.96	800
accuracy			0.94	20800
macro avg	0.94	0.94	0.94	20800
weighted avg	0.94	0.94	0.94	20800

Obr. 53 - Výsledek testování s využitím EMNIST testovacího sub datasetu (zdroj: vlastní)

3.4.2.2 Testování na vlastnoručně vytvořených vzorcích

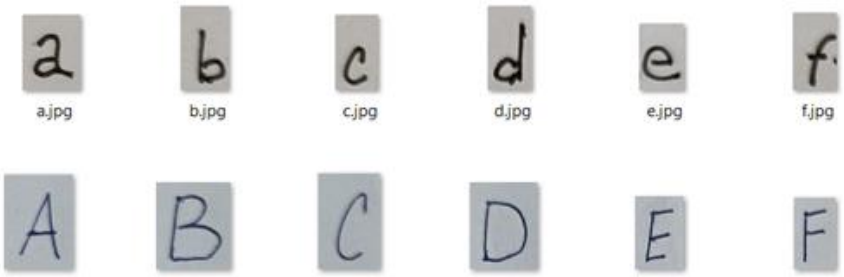
Soubor *testing_manager_external_images.py* využívá složku *assets*, ve které jsou uložené fotky vlastnoručně psaných písmen, které se neuronová síť snaží klasifikovat. Tuto

variantu je možné využít spuštěním souboru *main_offline.py*. Kompletní proces testování na vlastnoručně vytvořených datech znázorňuje následující vývojový diagram.



Obr. 54 - Vývojový diagram testovacího algoritmu vlastnoručně vytvořených dat (zdroj: vlastní)

Algoritmus po výběru datasetu spustí cyklus nad kolekcí v Pythonu označovanou jako slovník (dictionary), což je datová struktura složená z dvojic klíč: hodnota. Každý klíč je unikátní a je spojen s konkrétní hodnotou (číslo, řetězec, další slovník atd.). Tento konkrétní slovník mapuje písmena abecedy (hodnoty) k číselným hodnotám v rozsahu 0–25, které představují jednotlivé třídy, které je model neuronové sítě schopný klasifikovat. Cyklus má tak k dispozici v každé iteraci konkrétní písmeno a jeho číselnou reprezentaci. Tento cyklus se provede dvakrát, protože adresář *assets* obsahuje obrázky malých i velkých písmen pro přesnější porovnání (1 kus od každého písmene, dohromady 52 obrázků ke klasifikaci).



Obr. 55 - Ukázka vlastnoručně psaných písmen (zdroj: vlastní)

V rámci cyklu se poté vezme obrázek a předzpracuje se. Následně se jeho velikost upraví pomocí funkce knihovny *Numpy* do rozměrů 28x28 pixelů, což jsou rozměry, které očekává neuronová síť, neboť vzorky EMNIST datasetu mají stejné rozměry. V prvotní fázi testování, když byl využíván dataset ve variantě *ByClass* bylo nutno ještě přidat operaci, která obrázek před vstupem do neuronové sítě transponovala (rovněž z důvodu orientace obrázků v tomto datasetu). V dalším kroku dojde ke klasifikaci obrázku, kdy model sítě s využitím funkce *predict()* vrátí celočíselnou hodnotu reprezentující konkrétní třídu, do jaké obrázek klasifikoval. Pokud se toto číslo shoduje s aktuální číselnou hodnotou klasifikovaného obrázku, pak se inkrementuje proměnná *success*, v opačném případě je inkrementována proměnná *fail*. Jednotlivé výsledky se pomocí modulu Logger ukládají do .log souborů, kde jsou kromě jednotlivých výsledků typu úspěch / neúspěch tak procentuálně vyjádřené pravděpodobnosti, s jakou model určil, že se jedná o dané písmeno. Po ukončení cyklu dojde k výpisu výsledku, kde je úspěšnost konkrétního modelu sítě určena jako počet správných odhadů ku všem odhadům.

Tato metoda slouží jako zjednodušená metrika, jak mezi sebou porovnat přesnost jednotlivých modelů. Aby byly výsledky více objektivní, bylo by ideální mít v každé třídě větší množství ručně psaných vzorů, které by síť mohla klasifikovat. Celkové porovnání by tak bylo více robustní a vypovídající. Jelikož je ale stejná metoda využita na všechny modely pokaždé stejně a za stejných podmínek, lze tento způsob porovnání označit za vypovídající.

3.4.3 Instalace závislostí a spuštění projektu

Pro jednodušší implementaci neuronových sítí a jejich učení byla využita populární knihovna Tensorflow (poslední verze 2.13) podporující trénování na GPU. Jedná se o knihovnu napsanou v jazyce C++, která má vystavěné API pro Python, díky kterému může Python využívat rychlý C++ kód. Tensorflow ale od verze 2.10 nepodporuje trénování neuronových sítí na GPU nativně pod operačním systémem Windows. Trénování s využitím GPU je možné pouze pod Linuxovou distribucí. Pro trénování je možné využít i CPU, ale oproti GPU je to velmi neefektivní způsob, protože současné grafické karty jsou optimalizovány pro operace s plovoucí desetinnou čárkou, což je běžný typ výpočtu u trénování neuronových sítí, a to zejména při provádění násobení matic, jejich sčítání a jiných algebraických operací. Tento problém byl vyřešen využitím WSL2 („Windows Subsystem for Linux“).

3.4.3.1 WSL2 a konfigurace Tensorflow v prostředí Linuxové distribuce

Jedná se o kompatibilní vrstvu pro běh binárních souborů Linuxu v prostředí operačního systému Windows. Uživatel má tak možnost naistalovat konkrétní Linuxovou distribuci (Ubuntu, Debian, Fedora a další) a přistupovat k ní skrze příkazovou řádku a současně má tak daná distribuce možnost přistupovat k souborovému systému Windows, což umožňuje jednoduché sdílení dat. Není tak nutná kompletní virtualizace prostředí, což zvyšuje výkon a snižuje režii. Postup zprovoznění WSL2 je dostupný v (Loewen, 2023). Následně je nutné přihlásit se do WSL2 skrze vytvořený uživatelský účet a zapsat následující příkazy do konzole (aktuální adresář může uživatel zvolit libovolně):

1. *conda install -c conda-forge cudatoolkit=11.8.0;*
2. *python3 -m pip install nvidia-cudnn-cu11==8.6.0.163 tensorflow==2.13.*;*
3. *mkdir -p $CONDA_PREFIX/etc/conda/activate.d;*
4. *echo 'CUDNN_PATH=$(dirname $(python -c "import nvidia.cudnn;print(nvidia.cudnn.__file__)"))' >> $CONDA_PREFIX/etc/conda/activate.d/env_vars.sh;*
5. *echo 'export LD_LIBRARY_PATH=$CONDA_PREFIX/lib/:$CUDNN_PATH/lib:$LD_LIBRARY_PATH' >> $CONDA_PREFIX/etc/conda/activate.d/env_vars.sh;*
6. *source $CONDA_PREFIX/etc/conda/activate.d/env_vars.sh;*
7. *# Verify install;*
8. *python3 -c "import tensorflow as tf; print(tf.config.list_physical_devices('GPU'))";*

Tato sada příkazů využije správce balíčků Conda k nainstalování NVIDIA CUDA Toolkitu, což je sada nástrojů potřebných pro běh aplikací na platformě CUDA, což je platforma rovněž od společnosti NVIDIA, která umožňuje vývojářům využívat NVIDIA GPU pro výpočetní účely. Následně se už jako Python balíček nainstaluje NVIDIA cuDNN, což je

knihovna, která poskytuje vysoce optimalizované implementace primitivních funkcí používaných v neuronových sítích. Po následné konfiguraci cest by se po zadání posledního příkazu měl objevit následující řádek: *[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]* indikující, že došlo ke správnému nainstalování ovladačů a je možné využívat GPU pro trénování neuronových sítí. Kompletní postup je dostupný v (Tensorflow.org, 2023).

3.4.3.2 Virtuální prostředí, instalace závislostí a spuštění projektu

Jakmile je Tensorflow správně nakonfigurován, je možné doinstalovat zbývající závislosti. Korektním přístupem je vytvořit virtuální prostředí v rámci Pythonu a nainstalovat závislosti do něj, čímž dojde k izolaci závislostí od globálních balíčků, které využívá Python.

1. V rozhraní příkazové řádky je potřeba se přepnout do kořenového adresáře projektu a zadat příkaz: python3 -m venv myenv;
2. Následně je nutné aktivovat virtuální prostředí příkazem: myenv/Scripts/activate (v případě Windows) nebo source myenv/bin/activate (v případě Linuxu);
3. V posledním bodě se instaluje seznam závislostí příkazem: pip install -r requirements.txt.

Jakmile se dostahují potřebné závislosti, projekt bude možné spustit. To je možné buď spuštěním souboru z prostředí IDE nebo zadáním následujícího příkazu do příkazové řádky: „*uvicorn main_online:app --host 127.0.0.1 –port 8000* “. Tímto se spustí webový server na ip adrese 127.0.0.1:8000, což je adresa, se kterou komunikuje klient. Samotné webové rozhraní je pak dostupné na port :5173, jak je popsáno v kapitole 4.3.6 . Pro vývoj aplikace bylo užitečné k výše uvedenému příkazu ještě přidat „*--reload*“. Tím se docílí toho, že při každé změně kódu se webový server automaticky restartuje, aby bylo možné provedené změny ihned otestovat.

4 NÁVRH A REALIZACE KLIENTSKÉ ČÁSTI APLIKACE

Pro vytvoření grafického webového rozhraní je využita JavaScript knihovna React v kombinaci s TypeScriptem. TypeScript je programovací jazyk vyvinutý společností Microsoft jako nadstavba JavaScriptu. Veškerý platný JavaScript kód je zároveň platným TypeScript kódem. Naopak ne veškerý TypeScript kód je validním JavaScript kódem a během runtimu aplikace se Typescript trans-kompiluje do JavaScriptu. TypeScript doplňuje JavaScript o kontrolu datových typů (ta je vynucována během vývoje, ale ne během běhu aplikace), což je klíčová věc k odchycení chyb během vývoje předtím, než je kód nasazen. Aplikace dále využívá grafický framework MaterialUI, který definuje vzhled jednotlivých prvků a zajišťuje jednolitý vzhled stránky.

4.1 ZVOLENÉ TECHNOLOGIE

4.1.1 JavaScript

JavaScript je interpretovaný vysokoúrovňový programovací jazyk, který slouží jako základní stavební prvek každého současného interaktivního webu. Pokud nějaká webová stránka obsahuje tlačítka nebo vstupní pole, kam může uživatel zapsat nějakou hodnotu, tak je to právě JavaScript, který slouží k obsluze a zpracování těchto dat a jejich případné odeslání na server. Krom spuštění v prohlížeči, k čemuž je tento jazyk primárně určen, je v dnešní době běžnou praxí jeho využití i na straně serveru se speciálním prostředím jako Node.js. JavaScript byl vytvořen v roce 1995 americkým programátorem Brendanem Eichem. Správu a standardizaci JavaScriptu provádí organizace ECMA International pod názvem ECMAScript. Termín ECMAScript je standartní název jazyka, zatímco JavaScript je nejznámější implementací tohoto standardu. Jednotlivé verze ECMAScriptu se označují zkratkou ES3, ES5, ES6 atd (Čápka, 2013).

Vzhledem k popularitě tohoto jazyka vzniklo velké množství frameworků a knihoven, které definují různé přístupy k tvorbě webových aplikací. Mezi nejznámější a nejpoužívanější frameworky patří následující: React, Vue, Angular, Svelte, jQuery a mnoho dalších. Je nutno zmínit, že jednotlivé frameworky neovlivňují to, jak se uživateli zobrazí konkrétní stránka, ale ovlivňují pouze způsob, jakým programátor přistupuje k tvorbě aplikace.

4.1.2 React a TypeScript

Aplikace využívá knihovnu React v kombinaci s TypeScriptem. React se nepovažuje za plnohodnotný framework kvůli nutnosti využití dalších knihoven pro tvorbu kompletní aplikace. Typescript je využit z důvodu, že pomáhá odstranit hlavní nevýhodu JavaScriptu, a tou je jeho dynamická povaha, která neumožňuje statickou kontrolu typů. React je vyvíjen firmou Meta (dříve Facebook) s přispíváním open-source komunity a je nejpopulárnější ze zmíněných frameworků. To z něj činí ideálního kandidáta na tvorbu jakékoliv webové aplikace, protože ohledně něj existuje velké množství návodů a zdrojů informací. Fakt, že je React knihovnou, a ne plnohodnotným frameworkem, poskytuje programátorovi velkou volnost v přístupu k tvorbě aplikace, oproti například Angularu, který přesně definuje způsoby a praktiky, jak má být architektura aplikace strukturována (Cocca, 2023).

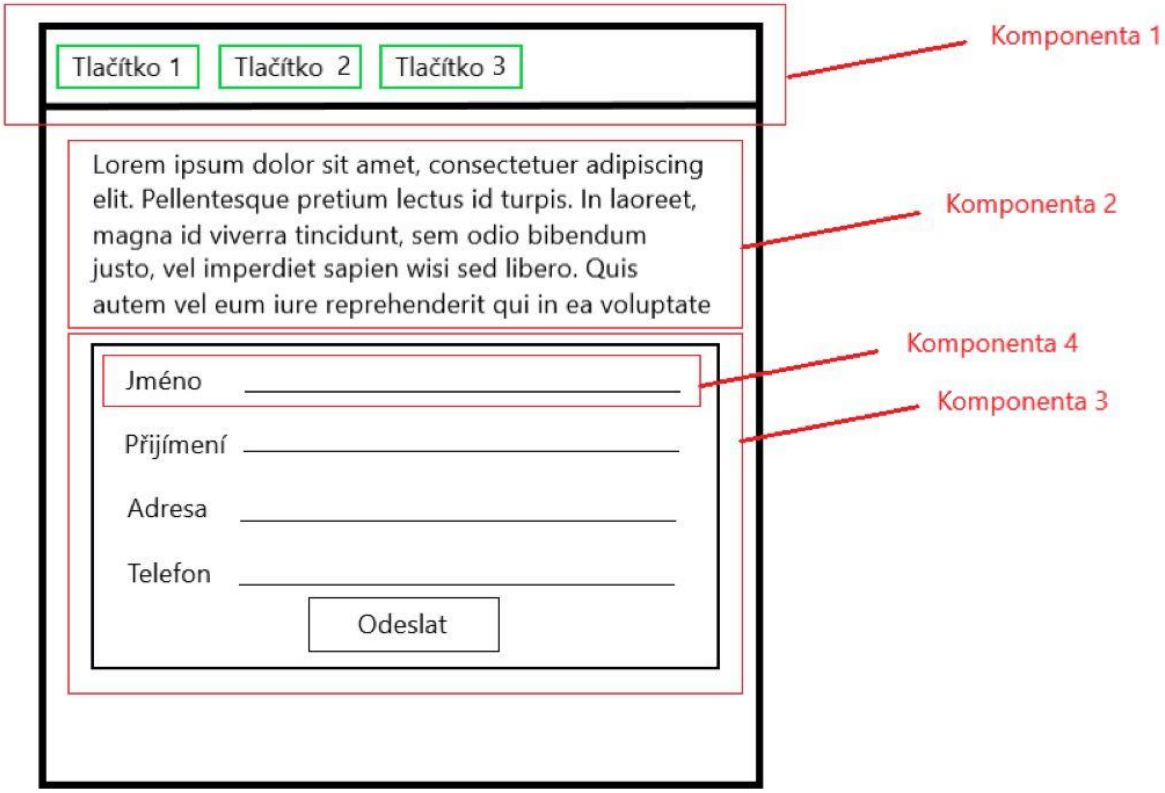
4.1.2.1 JSX/TSX

React nevyužívá klasické HTML soubory v kombinaci s JS/TS soubory jako ostatní frameworky. Místo toho využívá vlastní typ souborů označených jako JSX, což je zkratka pro JavaScript XML (TSX v případě využití TypeScriptu). Tento typ souboru tak umožňuje vkládat HTML kód přímo do JavaScriptu. Na první pohled může tento soubor připomínat klasický HTML, protože umožňuje definovat elementy pomocí tagů, atributů a potomků. Nejedná se však o HTML ani JavaScript. Kód JSX/TSX se tak překládá během runtimu do validního JavaScriptu pomocí trans-kompilátoru jako je Babel (Máca, 2019).

4.1.2.1 Přístup Reactu k tvorbě aplikace

React uplatňuje přístup rozdělení kódu na jednotlivé komponenty. Touto komponentou je možno si představit například vstupní pole ve formuláři, celý formulář nebo celou stránku, na které se formulář nachází. Komponenta může být prezenční („dummy“), ta pouze zobrazuje data, co jí jsou poskytnuta. Nebo může být inteligentní, to znamená, že udržuje stav aplikace a je zodpovědná za složitější operace (API volání). Celkové uživatelské rozhraní (UI) si tak lze představit jako strom komponent, kde každá komponenta odpovídá určité části uživatelského rozhraní. Taková reprezentace struktury se označuje jako DOM („Document Object Model“). React využívá koncept tzv. virtuálního DOMu. Je to (paměťově) lehká reprezentace skutečného DOMu a slouží k optimalizaci aktualizace skutečného DOMu v prohlížeči. React tak při změně stavu aplikace nejdříve aktualizuje virtuální DOM, následně algoritmem porovná rozdíly mezi

virtuálním a skutečným DOMem a aktualizuje jen ty části, kde došlo ke změně. To razantním způsobem zvyšuje rychlost a plynulost aplikace (Máca, 2019).



Obr. 56 – Ukázka možného rozložení UI na jednotlivé komponenty (zdroj: vlastní)

4.2 ARCHITEKTURA KLIENTSKÉ ČÁSTI APLIKACE

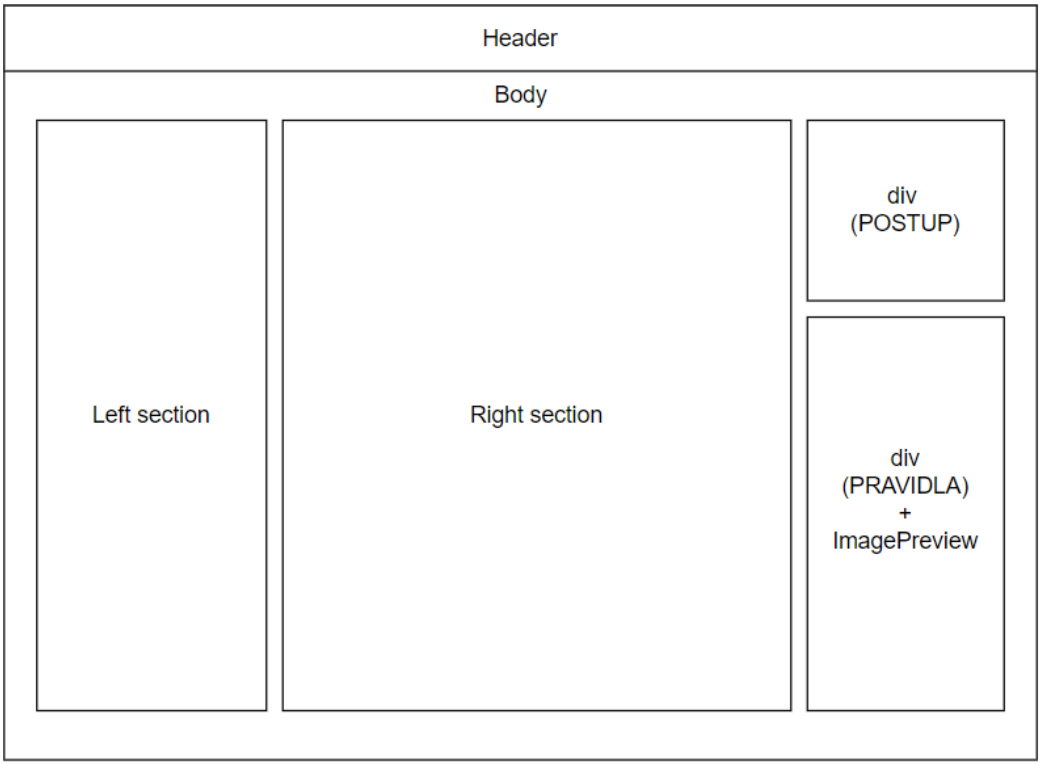
Aplikace je vytvořena jako tzv. SPA („Single Page Application“). Jedná se o standartní moderní přístup k tvorbě webových aplikací, kdy se běžně nevyužívá statických šablon, které by se do prohlížeče posílali ze serveru v moment, kdy chce uživatel přejít na další stránku. Místo toho je obsah generován dynamicky s využitím JavaScriptu, což vede k plynulejšímu pocitu responzivity aplikace. SPA obecně mohou mít více stránek, mezi kterýmiž je přechod implementován pomocí tzv. routeru. Nicméně v rámci této práce se jedná doslovně o aplikaci využívající jedinou stránku, čemuž odpovídá i jednoduší složková struktura.

Obr. 57 - Složková struktura klientské části aplikace (zdroj: vlastní)

Aplikace využívá funkcionální přístup s využitím tzv. React hooků, což jsou speciální metody, které se využívají v různých částech životního cyklu aplikace. Využívají se například k persistenci stavu, vytváření asynchronních API voláni a dalším operacím. Druhým možným přístupem je objektově orientovaný přístup, od kterého ale oficiální vývojářský tým Reactu upouští a doporučuje vývojářům využívat přístup funkcionální, který se snaží prosadit jako normu.

4.3 ROZLOŽENÍ UI A FUNKCE JEDNOTLIVÝCH KOMPONENT

Stránka je orientovaná na šířku, protože vývoj probíhal na monitoru s poměrem stran 21:9, je tedy problém vložit celý printscreen stránky, protože je pro zobrazení v dokumentu příliš široká. Z toho důvodu je vizualizována pomocí jednotlivých komponent, které jsou detailněji popsány v následujících kapitolách. Na obrázku č. 58 je ilustrované rozložení jednotlivých komponent, z obrázku č. 57.



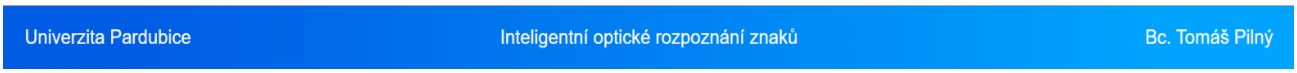
Obr. 58 - Rozložení UI (zdroj: vlastní)

4.3.1 Komponenta App.tsx

Toto je hlavní komponenta, která je importována v souboru *main.tsx*, což je hlavní vstup, odkud se aplikace načítá do prohlížeče. Komponentu tvoří jediný div tag, který je pomocí *SCSS* nastaven na celou šířku a výšku stránky. Tento div obsahuje dvě dětské komponenty *Header* a *Body*.

4.3.2 Komponenta Header.tsx

Tato jednoduchá prezenční komponenta slouží jako hlavička stránky. Skládá se z hlavního divu, ve kterém jsou vnořené další divy a odstavce.



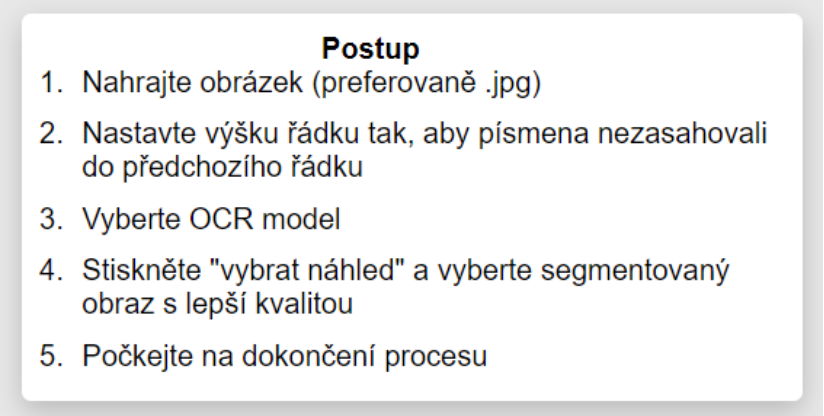
Obr. 59 - Komponenta Header.tsx (zdroj: vlastní)

4.3.3 Komponenta Body.tsx

Tato komponenta kombinuje jednotlivé komponenty zodpovědné za manipulaci s daty a obsluhu uživatelských vstupů. Je tvořena divem, který obsahuje komponenty *LeftSection*, *RightSection*, div (POSTUP) a div (PRAVIDLA), který sám ještě obsahuje komponentu *ImagePreview*. Zároveň je tato komponenta zodpovědná za persistenci dat výšky a posunu řádku a samotného obrázku a sdílí tuto informaci mezi komponentami *LeftSection* a *RightSection*.

4.3.3.1 Div (POSTUP)

V tomto divu je definován návod, jak aplikaci použít.



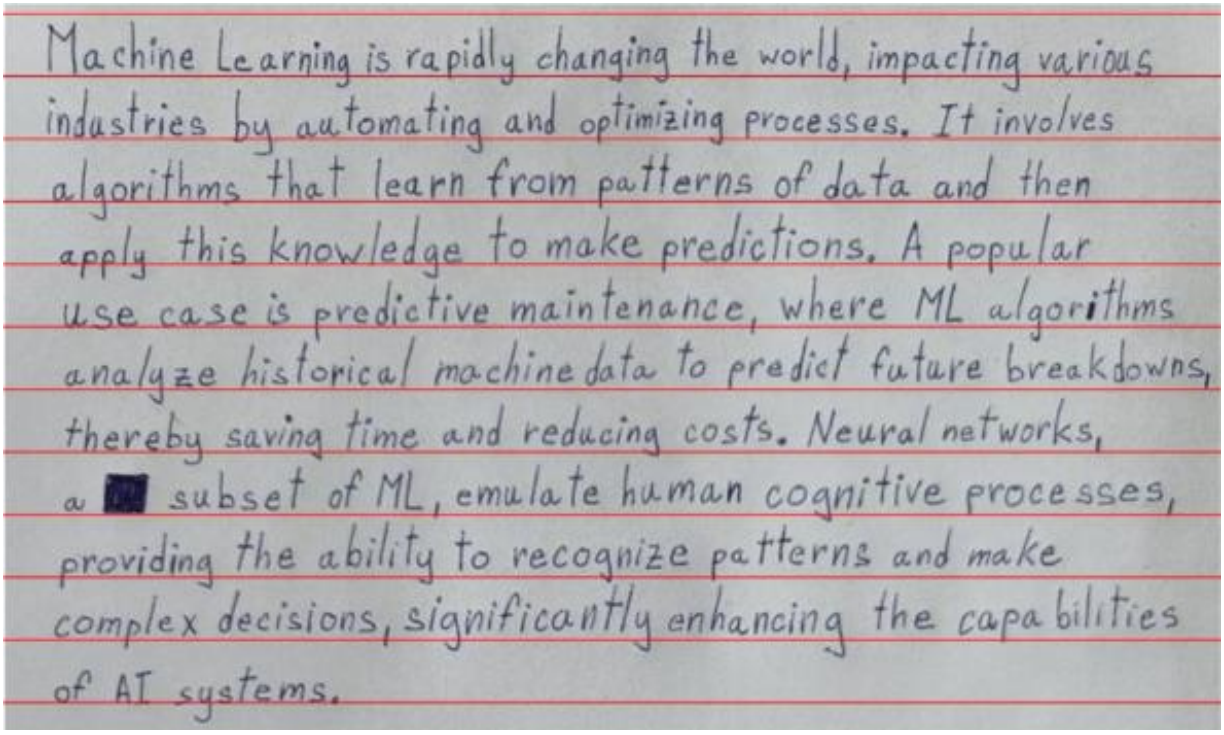
Obr. 60 - Návod na obsluhu aplikace (zdroj: vlastní)

4.3.3.2 Div (PRAVIDLA)

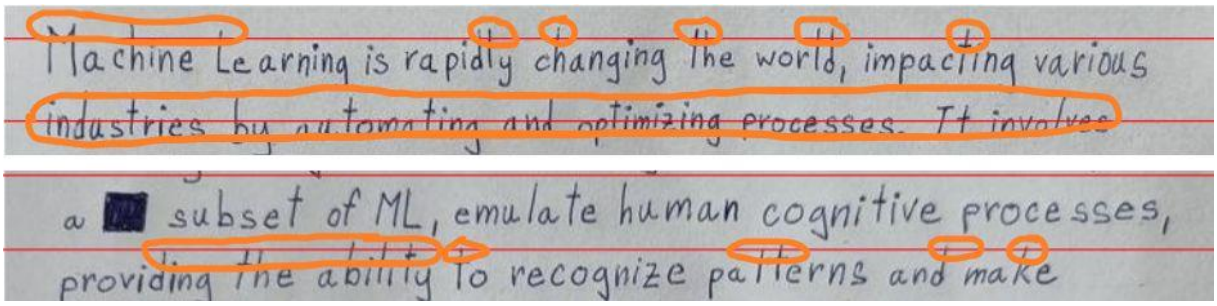
Tento div zobrazuje seznam kritérií uvedených v kapitole 3.3 , kterým musí vstupní obraz vyhovět, aby ICR mohlo být efektivně provedeno. Na konci divu je umístěno tlačítko, které zobrazí pop-up okno s komponentou *ImagePreview*.

4.3.3.3 Komponenta ImagePreview.tsx

Tato komponenta obsahuje dva vzorové obrázky ukazující, jak by měla vypadat vstupní fotografie (nasvětlení, orientace, ořez) a jak by u ní mělo být správně nastavené řádkování. Správné nastavení řádkování je důležité z toho důvodu, že pokud znak v rámci daného řádku (rozmezí mezi dvěma přímkami) zasahuje do řádku předchozího (prochází přímkou nad sebou), tak bude do tohoto řádku registrován, což naruší integritu ICR procesu. Naopak nevadí, pokud znak protíná přímku pod sebou.



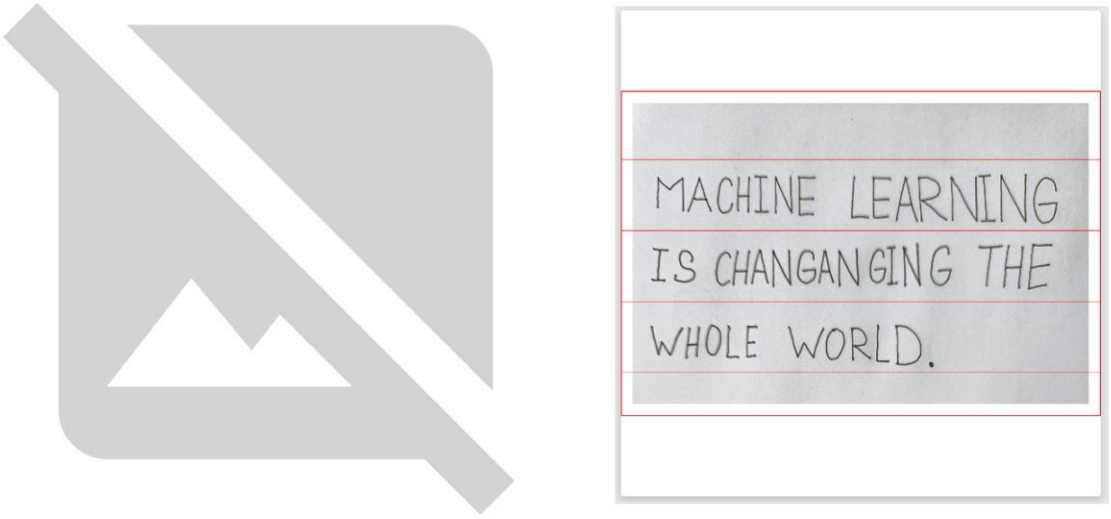
Obr. 61 - Ukázka správně nastaveného řádkování (zdroj: vlastní)



Obr. 62 - Ukázka špatně nastaveného řádkování, oranžová barva vyznačuje problémové části (zdroj: vlastní)

4.3.4 Komponenta RightSection.tsx

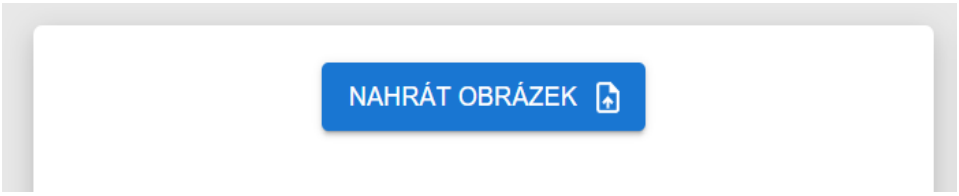
Tato komponenta slouží jako prezenční komponenta pro zobrazení obrázku s vyznačeným řádkováním. Ke kreslení přímek do obrázku se využívá knihovny *react-konva*. Když není přítomen obrázek, zobrazí se na šířku plochy komponenty ikona zobrazena na následujícím obrázku:



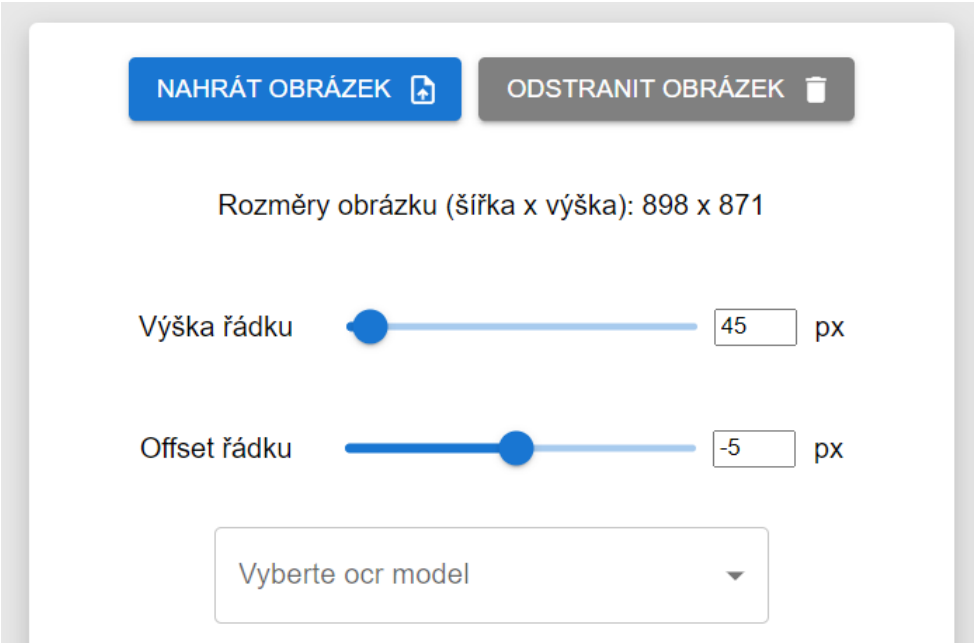
Obr. 63 - Komponenta RightSection bez obrazu (vlevo) a s obrazem (zdroj: vlastní)

4.3.5 Komponenta LeftSection.tsx

Tato komponenta slouží k nahrání obrázku na server pomocí API, které se volá po stisku tlačítka „Nahrát obrázek“, kde se upraví jeho velikost přidáním okrajů a následně se zobrazí v komponentě *RightSection.tsx*. Tato komponenta má dva různé vzhledy v závislosti na tom, jestli je právě nahraný obrázek nebo ne.

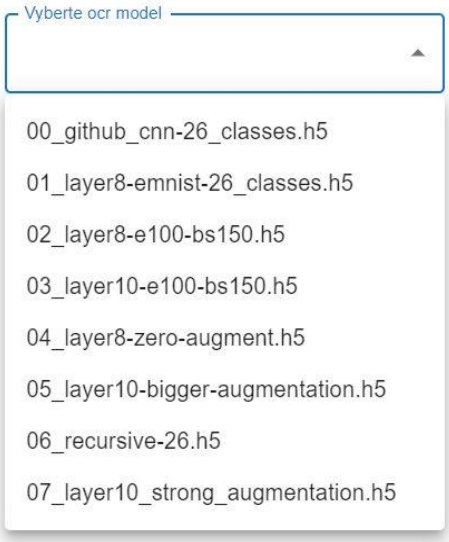


Obr. 64 - Komponenta LeftSection bez nahraného obrázku (zdroj: vlastní)



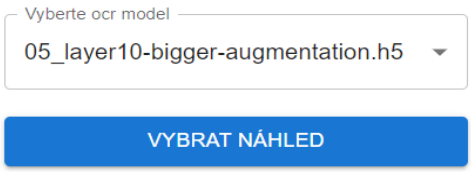
Obr. 65 - Komponenta LeftSection po nahrání obrázku (zdroj: vlastní)

Obrázek je možné v případě potřeby smazat, čímž se vzhled komponenty resetuje do podoby na obrázku č. 64. Uživatel má k dispozici dodatečnou informaci o rozměrech obrázku a pomocí sliderů může měnit vzdálenost mezi jednotlivými přímkami v obrazu a jejich posun v Y ose. Po rozkliknutí drop-down menu se provolá API („*/models*“) a zobrazí se seznam vytrénovaných modelů uložených na serveru ve složce *src/trained_models/emnist_dataset_26*.



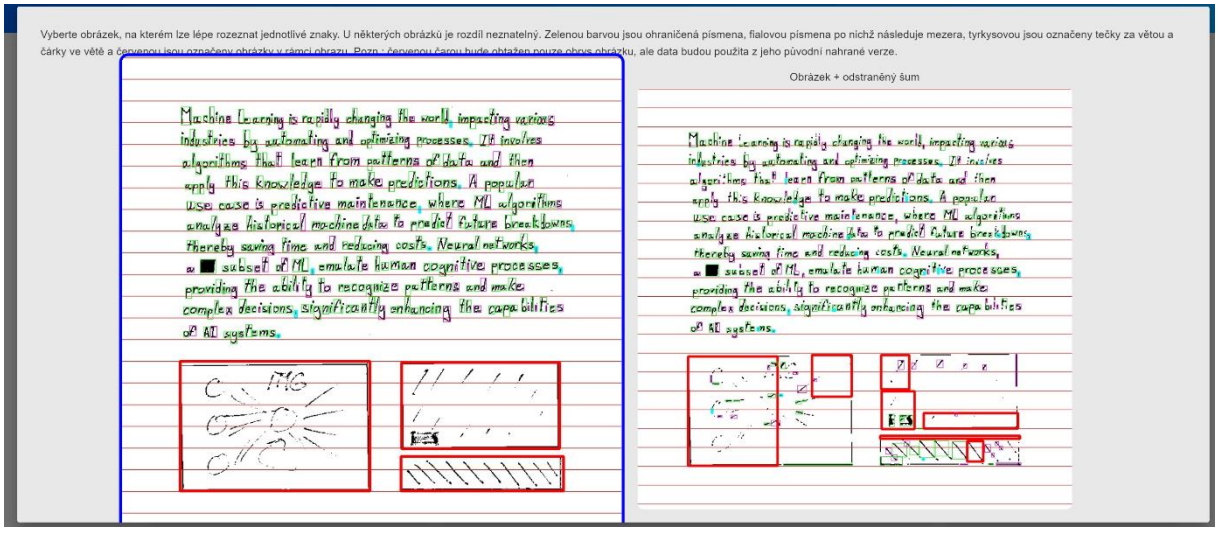
Obr. 66 - dostupné modely v moment pořizování snímku obrazovky (zdroj: vlastní)

Jakmile uživatel vybere jeden z modelů, seznam se skryje a objeví se pod ním tlačítko „Vybrat náhled“.



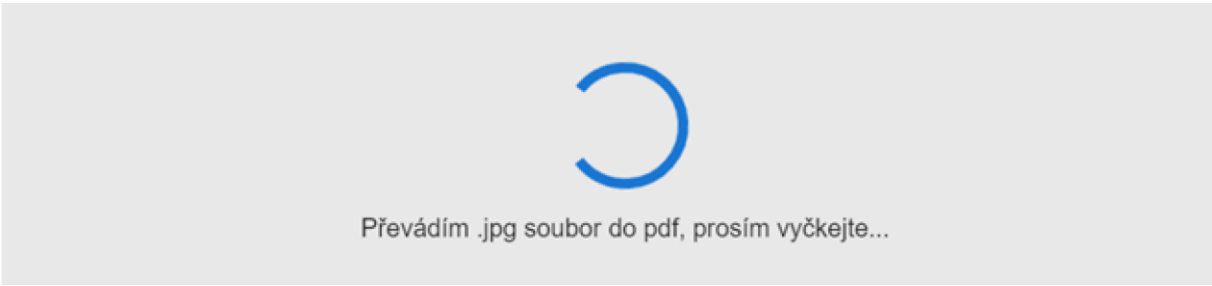
Obr. 67 - Tlačítko vybrat náhled (zdroj: vlastní)

Po stisknutí tohoto tlačítka se provolá API („*/preprocess*“) a na server se odešlou hodnoty výšky a posunu řádku. Server následně s využitím těchto informací provede segmentaci a vrátí klientovi dvě varianty binarizovaného obrazu (jak bylo zmíněno v kapitole 3.3.1). Uživateli se zobrazí pop-up okno, které obsahuje tyto dva obrázky jako elementy, na které lze kliknout. Pro lepší porovnání se obrázek po najetí myši o něco zvětší.



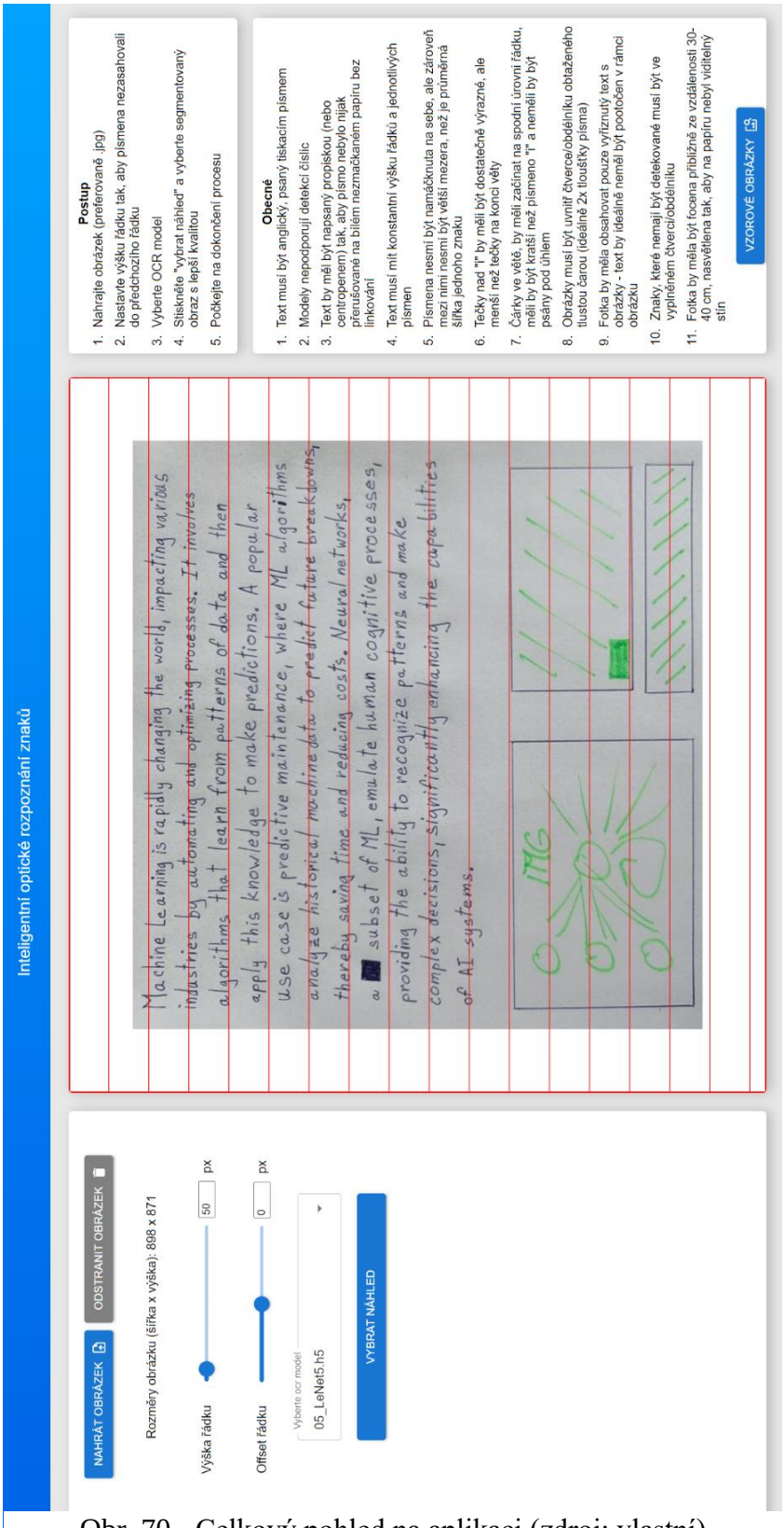
Obr. 68 - Zobrazení výběru obrázků určených ke klasifikaci (zdroj: vlastní)

Jakmile pak uživatel na zvolený obrázek klikne, provolá se API („*/start-ocr*“) a zobrazí se spinner s hláškou „Převádím .jpg soubor do pdf prosím vyčkejte…“



Obr. 69 - Načítací spinner během provádění ICR (zdroj: vlastní)

Po dokončení procesu se otevře systémové okno, kde už si uživatel vybere, kam chce uložit obdržený ZIP soubor. Tímto je proces ICR ukončen a uživatel může v případě potřeby nahrát další obrázek a celý postup rozebraný v předešlých kapitolách zopakovat.



Obr. 70 - Celkový pohled na aplikaci (zdroj: vlastní)

4.3.6 Spuštění aplikace

Projekt využívá správce balíčků pro JavaScript s názvem npm („node package manager“), s jehož využitím lze instalovat jednotlivé závislosti z prostředí příkazového řádku příkazem *npm install*. Tímto dojde k nainstalování všech závislostí uvedených v souboru *package.json.* Pro spuštění projektu se využije příkazu *npm run dev*, který spustí vlastní webový server na adrese 127.0.0.1:5173. Aplikace využívá nástroj pro sestavení webových aplikací („build tool“) s názvem Vite, který obsahuje i samotný vývojový server. Jeho autorem je Evan You, který je taktéž autorem frameworku Vue. V projektu je v kořenovém adresáři umístěn soubor s názvem *vite.config.ts*, kde je možné nastavit v případě potřeby jinou adresu a port, na kterých bude aplikace dostupná.

5 POROVNÁNÍ JEDNOTLIVÝCH MODELŮ

Tato kapitola porovnává průběhy trénování a výslednou přesnost a výkonost jednotlivých modelů neuronových sítí. Celkem bylo otestováno osm různých topologií sítí. Trénování probíhalo na grafické kartě RTX 4070Ti., kde průměrné vytížení CUDA jader dosahovalo mezi 50–80 % během učení. Tato hodnota se zvyšovala nebo snižovala v závislosti na složitosti modelu. K porovnání jsou využity obě varianty testovacího modulu zmíněného v kapitole 3.4.2 . Současně je přesnost všech modelů porovnána na konkrétní klasifikační úloze na obrazu, který je pro všechny modely stejný. Všechny modely byly trénovány s následujícími parametry:

- Počet epoch: 250;
- Batch size: 120;
- Augmentace:
  - Rozsah natočení obrazu: ±20 °;
  - Rozsah přiblížení/oddálení: 15 %;
  - Posun v ose X a ose Y: 15 %;
- Optimalizace: Adam (learning_rate=0.001);
- Chybová funkce: Kategorická křížová entropie;
- EarlyStopping(motior='val_loss', patience=13,restore_best_weights=True);
- ReduceLROnPlateau(monitor='val_loss',patience=6,factor=0.5,min_lr=1e-6).

5.1 UPRAVENÝ MODEL RESNET18

Tento model je upravenou verzí populárního modelu ResNet18 představeného v publikaci „Deep Residual Learning for Image Recognition" (Kaiming, 2015). Původní verze pracuje se vstupními daty o rozměrech 224x224x3 pixelů. Úpravy v této verzi ResNet18 napomáhají přizpůsobit model sítě klasifikační úloze s menšími vstupními daty. Síť se skládá z úvodní konvoluční vrstvy o rozměrech 3x3 s 64 filtry a krokem posunu („stride") 1. Tělo sítě se pak skládá z šesti tzv. reziduálních bloků. Každý reziduální blok je tvořen dvěma konvolučními vrstvami a zkratkou („shortcut connection"). Zkratka umožňuje, aby výstupní signál bloku byl kombinací původního vstupního signálu a změn (reziduí) z konvolučních vrstev. Jinými slovy, místo toho, aby se síť snažila přímo modelovat výstup, tak se snaží modelovat změnu k vstupnímu signálu. Zkratka tak síti umožňuje „nedělat nic" v případě, že se přidané konvoluční vrstvy projeví pro danou část trénování jako nepotřebné. Pokud je tak optimální výstup z bloku blízký jeho vstupu (ideální identita), pak konvoluční vrstvy mohou

vytvářet minimální nebo nulové reziduální hodnoty a zkratka tak může přenést původní signál skrze konvoluční vrstvu beze změny. To pomáhá odstranit problém mizejícího gradientu (zmíněno v kapitole 2.5.3), se kterým se potýkají modely hlubokých konvolučních neuronových sítí. Po reziduálních blocích strukturu sítě dokončuje kombinace globální AveragePooling vrstvy spolu s Dropout a Dense vrstvou.

5.2 UPRAVENÝ MODEL LENET5

Tento model byl navržen Yannem LeCunem v devadesátých letech minulého století. Jedná se o výrazně jednodušší model oproti ResNet18. Původní model LeNet5 se využíval pro klasifikaci ručně psaných číslic z databáze MNIST. Tato varianta je upravena pro klasifikace ručně psaných písmen a tudíž má 26 výstupních tříd. Další změnou je využití hyperbolicko-tangentní aktivační funkce oproti původní sigmoidní funkci. Posledním rozdílem je velikost vstupního obrazu, která v původní verzi byla 32x32x1 pixelů a v této variantě využívá 28x28x1 pixelů. Síť je tvořena dvojicí konvoluční vrstvy a AveragePooling vrstvy následována Flatten vrstvou a třemi vrstvami Dense.

5.3 MODEL KONVOLUČNÍ SÍTĚ – VARIANTA Č. 1

Jedná se o poměrně základní model konvoluční neuronové sítě, který je strukturou podobný modelu LeNet5. Je tvořen dvěma dvojicemi konvoluční a MaxPooling vrstvy s ReLu aktivační funkcí. Stejně jako v modelu LeNet5 následuje vrstva Flatten transformující výstupní matici na jednorozměrný vektor. Nakonec data projdou dvojicí Dense vrstev propojených Dropout vrstvou, která je nastavena tak, aby aktivovala pouze 50 % všech neuronů a tím se předešlo přeučení sítě.

5.4 MODEL KONVOLUČNÍ SÍTĚ – VARIANTA Č. 2

Tento model je velmi podobný předchozí variantě, má ale oproti ní pouze jednu MaxPooling vrstvu za účelem snížení odstranění informace z obrazu. Za touto vrstvou navíc následuje ještě Dropout vrstva nastavena na 25 %, aby síť byla robustnější proti přeučení.

5.5 MODEL KONVOLUČNÍ SÍTĚ – VARIANTA Č. 3

Tento model je z variant konvolučních sítí nejsložitější. Stejně jako první varianta využívá dvojice konvoluční a MaxPooling vrstvy, ale oproti první variantě se skládá ze tří těchto dvojic. Zbytek sítě se shoduje s druhou polovinou druhé varianty, to znamená kombinace vrstev Dropout, Flatten, Dense, Dropout a poslední vrstvou Dense s aktivační funkcí softmax.

5.6 UPRAVENÝ MODEL VGGNET16

Tento model je modifikací populárního modelu VGGNet-16 představeným týmem z Oxfordské univerzity v roce 2013 (Simonyan, Zisserman, 2014). Původní model využívá 16 vrstev (13 konvolučních a 3 Dense) ke klasifikaci obrazů z datasetu ImageNet, který je tvořen vzorky obrazů o rozměrech 224x224x3 pixelů. Jedná se tak o další model konvoluční neuronové sítě, který je upraven tak, aby topologie sítě odpovídala klasifikační úloze ručně psaných písmen o rozměrech 28x28x1 pixelů. Síť je tvořena dvěma bloky složenými ze dvou konvolučních vrstev s rozměry filtrů 3x3 MaxPooling a Dropout vrstvy. Po těchto blocích následuje Flatten vrstva a dvě dvojice vrstev Dense a Dropout. Poslední vrstvou jako v ostatních případech je Dense vrstva s 26 neurony a aktivační funkcí softmax.

5.7 UPRAVENÝ MODEL MOBILENET

Tento model je inspirován architekturou MobileNet navrženou pro mobilní a vestavěné aplikace týmem vývojářů z firmy Google (Howard, 2017). MobileNet využívá tzv. „Depthwise separable convolution“ místo standartních konvolučních vrstev. Takto definované konvoluční vrstvy snižují počet výpočetních operací a zároveň zachovávají kvalitu modelu. Primární myšlenka originálního modelu je být výpočetně efektivní, aby mohl být nasazen na zařízení s omezenými výpočetními zdroji, jako jsou mobilní telefony. Stejně jako originální varianta VGGNet-16 se i tento model používá pro klasifikaci obrazů z ImageNet datasetu. Tato upravená verze je opět přizpůsobena pro velikost obrazu a počet výstupních tříd. Jedná se o složitější konvoluční model tvořený na vstupu klasickou konvoluční vrstvou spolu s BatchNormalization vrstvou. Tato vrstva mění výstupy předchozí vrstvy tak, aby měly nulovou střední hodnotu a jednotkovou standartní odchylku. To může vést k rychlejší konvergenci a trénovací rychlosti. Následuje sedm bloků, kde se každý blok skládá z kombinace Depthwise konvoluční vrstvy, BatchNormolization vrstvy, konvoluční vrstvy a další BatchNormalization vrstvy. Síť je zakončena AveragePooling vrstvou spolu s Dropout (nastavenou na 50 %) a Dense vrstvou se softmax aktivační funkcí.

5.8 MODEL REKURENTNÍ SÍTĚ

Posledním využitým modelem pro srovnání je rekurentní neuronová síť skládající se z pěti vrstev. První vrstva Reshape přeformátuje vstupní data o rozměrech 28x28x1 pixelů na 28 sekvencí o délce 28 pixelů. Následují dvě vrstvy LSTM, kde první z nich vrací sekvence do druhé vrstvy, která je zpracovává. Následuje Dropout vrstva nastavená na 50 % a Dense vrstva s 26 neurony a softmax aktivací.

5.9 TOPOLOGIE MODELŮ

```
def build_custom_resnet18(input_shape=(28, 28, 1), num_classes=26,
                          regularizer_size=0.001) -> Model:
    inputs = tf.keras.Input(shape=input_shape)
    x = layers.Conv2D(64, (3, 3), strides=(1, 1), padding='same',
                      kernel_regularizer=regularizers.l2(regularizer_size))(inputs)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)
    x = _residual_block(x, 64)
    x = _residual_block(x, 64)
    x = _residual_block(x, 128, stride=2, regularizer_size=regularizer_size)
    x = _residual_block(x, 128)
    x = _residual_block(x, 256, stride=2, regularizer_size=regularizer_size)
    x = _residual_block(x, 256)
    x = layers.GlobalAveragePooling2D()(x)
    x = layers.Dropout(0.5)(x)
    outputs = layers.Dense(num_classes, activation='softmax',
                           kernel_regularizer=regularizers.l2(regularizer_size))(x)
    model = models.Model(inputs, outputs)
    print(model.summary())
    return model


def _residual_block(x, filters, stride=1, regularizer_size=0.0001):
    shortcut = x
    x = layers.Conv2D(filters, (3, 3), strides=stride, padding='same',
                      kernel_regularizer=regularizers.l2(regularizer_size))(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)
    x = layers.Conv2D(filters, (3, 3), strides=1, padding='same',
                      kernel_regularizer=regularizers.l2(regularizer_size))(x)
    x = layers.BatchNormalization()(x)
    if stride != 1 or int(shortcut.shape[-1]) != filters:
        shortcut = layers.Conv2D(filters, (1, 1), strides=stride,
                                 kernel_regularizer=regularizers.l2(regularizer_size))(shortcut)
        shortcut = layers.BatchNormalization()(shortcut)
    x = layers.Add()([shortcut, x])
    x = layers.Activation('relu')(x)
    return x
```

Obr. 71 - Struktura modifikované sítě ResNet18 (zdroj: vlastní)

```python
def build_lenet5(input_shape=(28, 28, 1), num_classes=26):
    model = models.Sequential([
        layers.Conv2D(6, kernel_size=(5, 5), activation='tanh',
                      input_shape=input_shape, padding='same'),
        layers.AveragePooling2D(pool_size=(2, 2)),
        layers.Conv2D(16, kernel_size=(5, 5), activation='tanh'),
        layers.AveragePooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dense(120, activation='tanh'),
        layers.Dense(84, activation='tanh'),
        layers.Dense(num_classes, activation='softmax')
    ])
    print(model.summary())
    return model
```

Obr. 72 - Struktura modifikované sítě LeNet5 (zdroj: vlastní)

```python
def build_cnn_topology_v1(input_shape=(28, 28, 1), num_classes=26):
    model = models.Sequential([
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation='softmax')
    ])
    print(model.summary())
    return model
```

Obr. 73 - Struktura konvoluční sítě – varianta č. 1 (zdroj: vlastní)

```python
def build_cnn_topology_v2(num_classes=26) -> Sequential:
    model = models.Sequential([
        layers.Conv2D(32, kernel_size=(3, 3), activation='relu',
                      input_shape=(28, 28, 1)),
        layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Dropout(0.25),
        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation='softmax')
    ])
    print(model.summary())
    return model
```

Obr. 74 - Struktura konvoluční sítě – varianta č. 2 (zdroj: vlastní)

```python
def build_cnn_topology_v3() -> Sequential:
    model = Sequential()
    model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)))
    model.add(MaxPool2D(pool_size=(2, 2), strides=2))
    model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same'))
    model.add(MaxPool2D(pool_size=(2, 2), strides=2))
    model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='valid'))
    model.add(MaxPool2D(pool_size=(2, 2), strides=2))
    model.add(Dropout(0.25))
    model.add(Flatten())
    model.add(Dense(128, activation="relu"))
    model.add(Dropout(0.5))
    model.add(Dense(26, activation="softmax"))
    print(model.summary())
    return model
```

Obr. 75 - Struktura konvoluční sítě – varianta č. 3 (zdroj: vlastní)

```python
def build_vgg16():
    model = Sequential()
    model.add(Conv2D(64, (3, 3), padding='same', activation='relu', input_shape=(28, 28, 1)))
    model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.1))
    model.add(Conv2D(128, (3, 3), padding='same', activation='relu'))
    model.add(Conv2D(128, (3, 3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.1))
    model.add(Flatten())
    model.add(Dense(4096, activation='relu'))
    model.add(Dropout(0.25))
    model.add(Dense(4096, activation='relu'))
    model.add(Dropout(0.25))
    model.add(Dense(26, activation='softmax'))
    print(model.summary())
    return model
```

Obr. 76 - Struktura modifikované sítě VGGNet-16 (zdroj: vlastní)

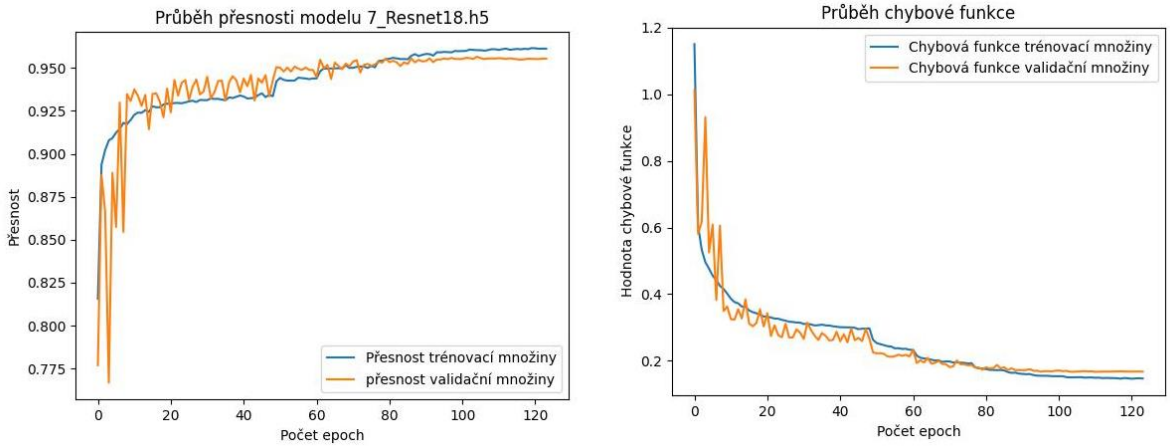
```python
def build_mobilenet(input_shape=(28, 28, 1), num_classes=26):
    inputs = tf.keras.Input(shape=input_shape)
    x = Conv2D(filters=32, kernel_size=3, strides=1, padding='same')(inputs)
    x = BatchNormalization()(x)
    x = ReLU()(x)
    x = mobilenet_block(x, filters=64, strides=1)
    x = mobilenet_block(x, filters=128, strides=2)
    x = mobilenet_block(x, filters=128, strides=1)
    x = mobilenet_block(x, filters=256, strides=2)
    x = mobilenet_block(x, filters=256, strides=1)
    x = mobilenet_block(x, filters=512, strides=2)
    x = mobilenet_block(x, filters=512, strides=1)
    x = GlobalAveragePooling2D()(x)
    x = Dropout(0.5)(x)
    output = Dense(num_classes, activation='softmax')(x)
    model = Model(inputs=inputs, outputs=output)
    return model


def mobilenet_block(x, filters, strides):
    x = DepthwiseConv2D(kernel_size=3, strides=strides, padding='same')(x)
    x = BatchNormalization()(x)
    x = ReLU()(x)
    x = Conv2D(filters=filters, kernel_size=1, strides=1, padding='same')(x)
    x = BatchNormalization()(x)
    x = ReLU()(x)
    return x
```

Obr. 77 - Struktura modifikované sítě MobileNet (zdroj: vlastní)

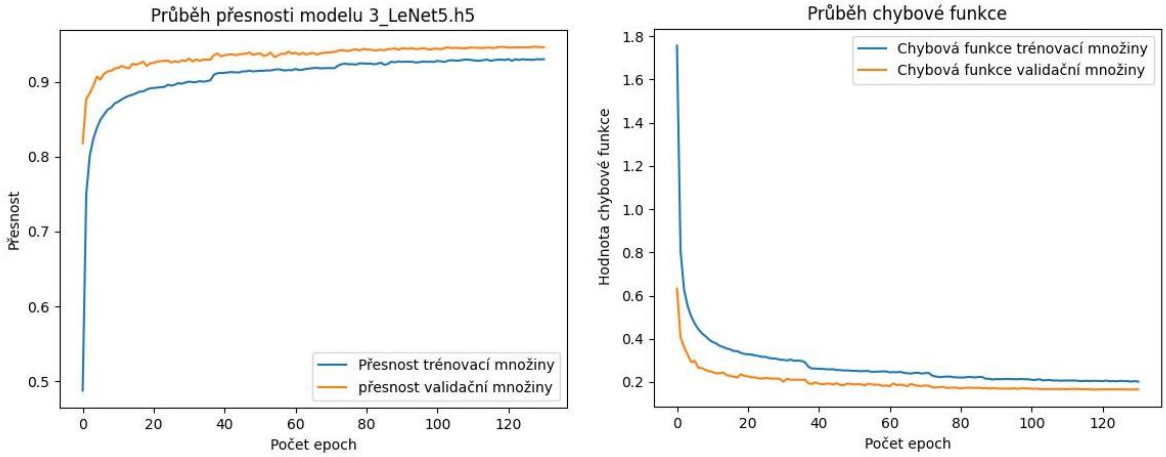
```python
def build_recursive_cnn(input_shape=(28, 28, 1), num_classes=26):
    model = models.Sequential([
        layers.Reshape((28, 28), input_shape=input_shape),
        layers.LSTM(128, activation='relu', return_sequences=True),
        layers.LSTM(128, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation='softmax')
    ])
    return model
```

Obr. 78 - Struktura rekurentní sítě (zdroj: vlastní)

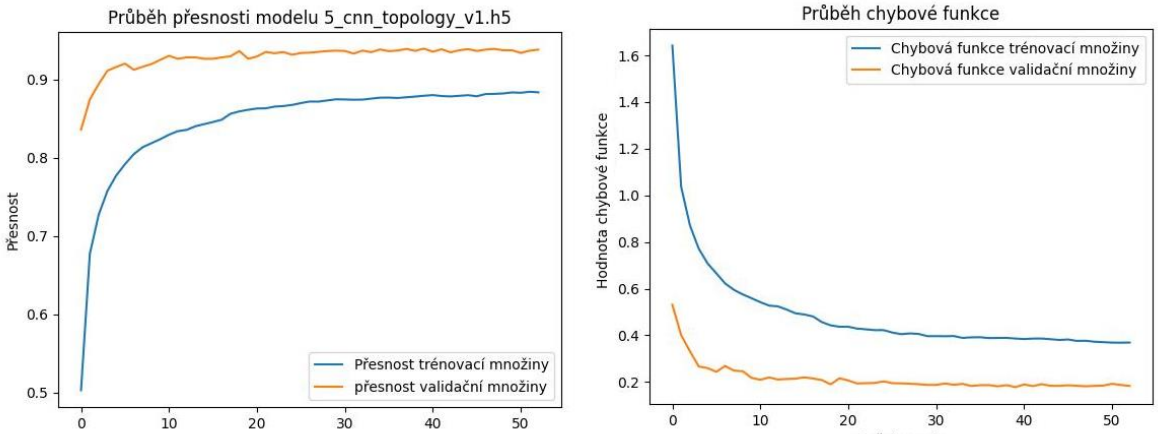
5.10 GRAFY PRŮBĚHŮ TRÉNOVÁNÍ



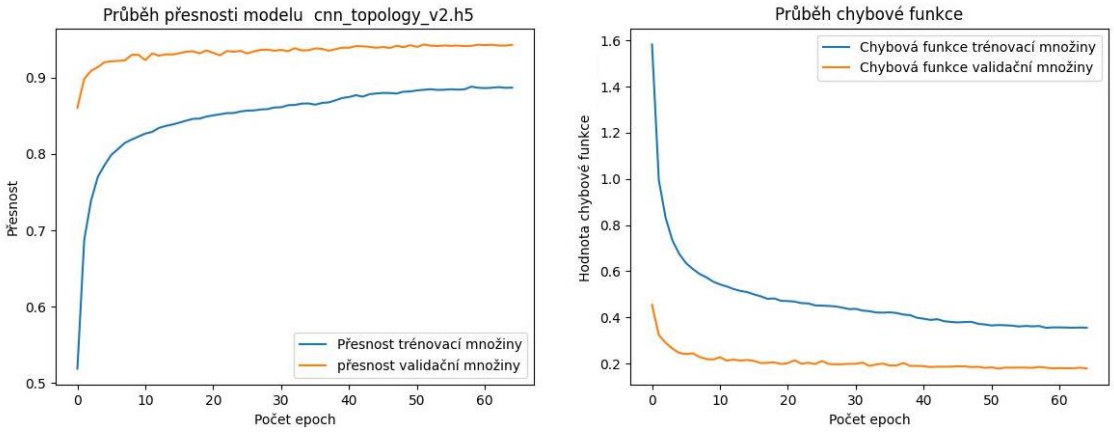
Obr. 79 - Průběh trénování modelu ResNet18 (zdroj: vlastní)



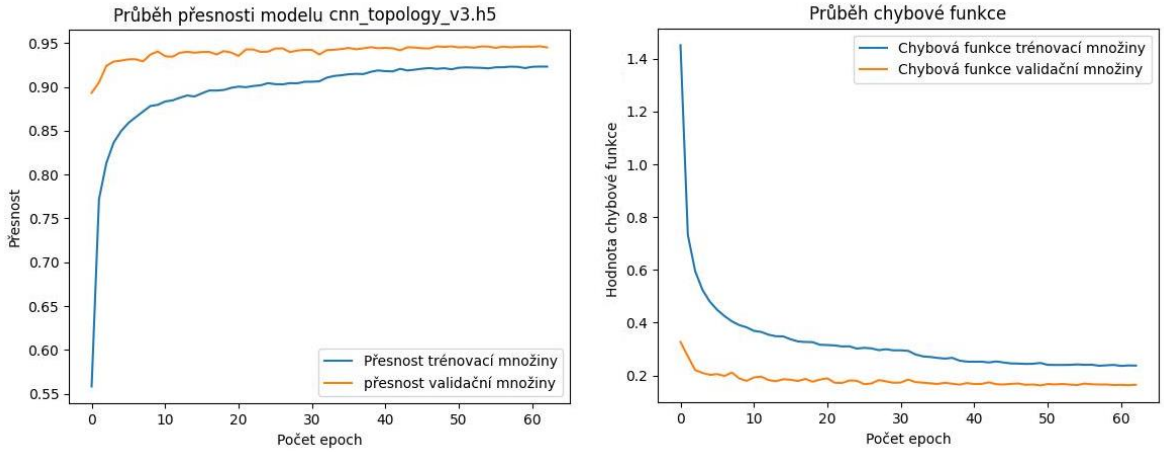
Obr. 80 - Průběh trénování modelu LeNet5 (zdroj: vlastní)



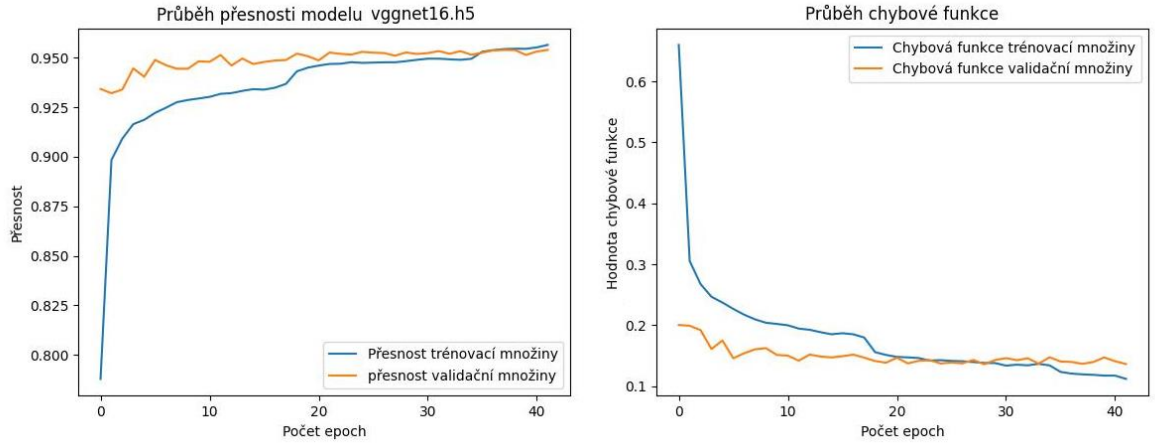
Obr. 81 - Průběh trénování konvoluční sítě – varianta č. 1 (zdroj: vlastní)



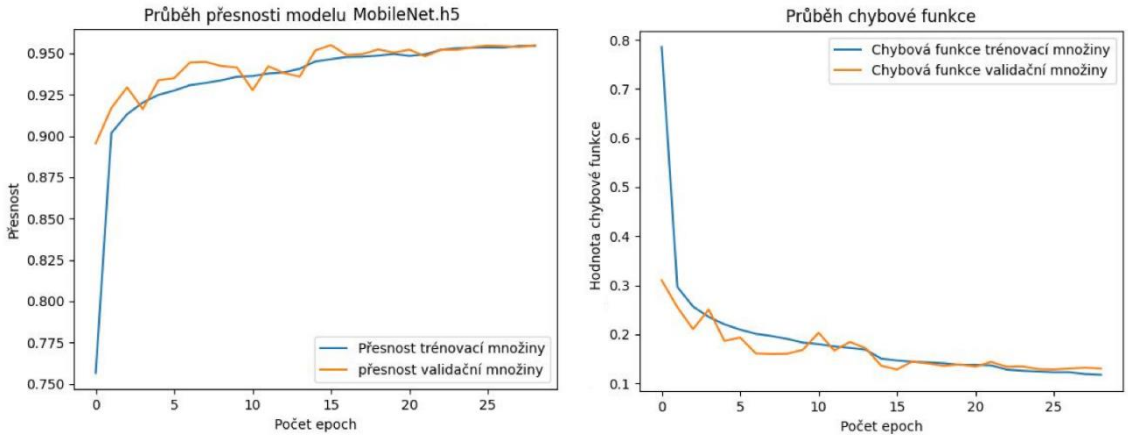
Obr. 82 - Průběh trénování konvoluční sítě – varianta č. 2 (zdroj: vlastní)



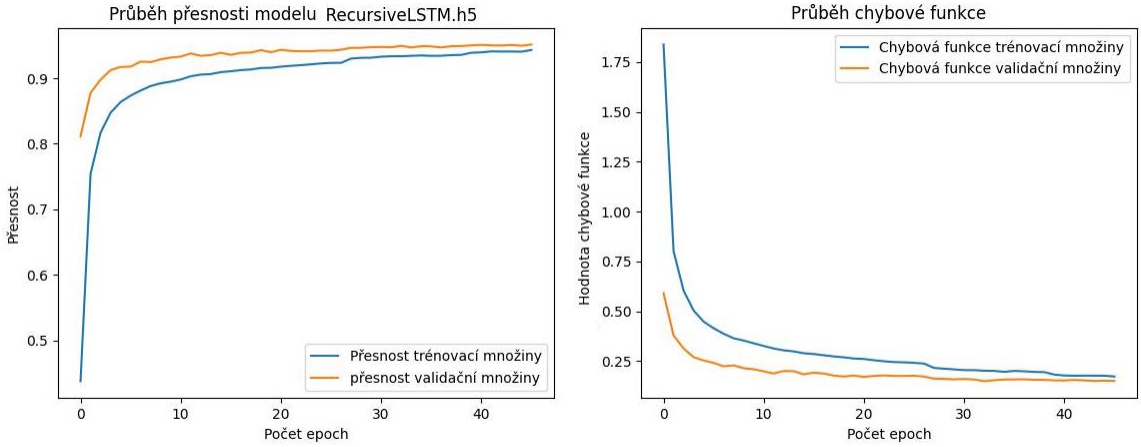
Obr. 83 - Průběh trénování konvoluční sítě – varianta č. 3 (zdroj: vlastní)



Obr. 84 - Průběh trénování modelu VGGNet-16 (zdroj: vlastní)



Obr. 85 - Průběh trénování modelu MobileNet (zdroj: vlastní)



Obr. 86 - Průběh trénování modelu rekurzivní sítě (zdroj: vlastní)

5.11 ČÍSELNÉ POROVNÁNÍ

5.11.1 Testovací dataset EMNIST

K prvnímu porovnání slouží testovací dataset EMNIST s funkcí *classification_report ()*, jak bylo uvedeno v kapitole 3.4.2.1.

Název modelu	*Testovací dataset EMNIST (classification_report()), počet vzorků: 20800*				
	Počet trénovatelných parametrů	precision	recall	f1-score	čas
ResNet18	2 782 874	95%	95%	95%	37.3s
LeNet5	63 066	94%	94%	94%	1.2s
CNN_V1	227 098	94%	94%	93%	1.5s
CNN_V2	1 201 946	94%	94%	94%	3.0s
CNN_V3	161 690	95%	95%	95%	2.1s
VGGNet-16	42 841 050	95%	95%	95%	14.1s
MobileNet	553 946	95%	95%	95%	10.9s
Rekurentní síť	215 322	95%	95%	95%	3.6s

Obr. 87 - Porovnání přesnosti modelů pomocí funkce classification_report (zdroj: vlastní)

Jak je vidět na obrázku č. 87, všechny modely na testovacím datasetu vykazují prakticky stejné výsledky. To lze vysvětlit jednak tím, že modely jsou dostatečně kvalitně vytvořeny a vytrénovány pro danou klasifikační úlohu. To ostatně naznačují i tvary a dosažené hodnoty grafů jednotlivých modelů. Tento fakt může též poukazovat na to, že testovací část datasetu může být jednodušší pro klasifikaci (například vzorky obsahují méně šumu). V opačném případě by tyto výsledky mohly ukazovat na přeučení modelů sítě, ale to jen v případě, že by testovací dataset nebyl oddělen od trénovacího.

Aby byl tento způsob porovnaní vypovídající vzhledem k výsledkům, je potřeba ho svázat s nějakým dalším kritériem. V tomto případě to může být doba zpracování testovacího datasetu o délce 20 800 vzorků. Za těchto podmínek nejlepšího výsledku dosáhla síť LeNet5 a nejhoršího výsledku síť ResNet18. Na první pohled se může zdát, že zde existuje přímá úměra mezi dobou testování a počtem trénovatelných parametrů, ale při pohledu na ostatní modely a jejich množství trénovatelných parametrů a výsledné časy lze vidět, že takováto úměra neplatí. V rámci tohoto testování bylo možné zjistit, že pro vybranou skupinu tříd lze pozorovat znatelně horší přesnost než pro zbytek tříd. Konkrétně se jedná o třídy odpovídající písmenům

G, I, L a O. Následující obrázek ilustruje nejnižší dosažené procentuální výsledky pro dané třídy na jednotlivých modelech.

Název modelu	*Testovací dataset EMNIST (classification_report()), počet vzorků: 20800*											
	precision (lowest)				recall (lowest)				f1-score (lowest)			
	G	I	L	Q	G	I	L	Q	G	I	L	Q
ResNet18	92%	79%	73%	87%	86%	69%	82%	93%	89%	74%	77%	90%
LeNet5	88%	76%	75%	85%	84%	73%	77%	88%	86%	74%	76%	86%
CNN_V1	88%	76%	73%	82%	80%	71%	77%	89%	84%	73%	75%	85%
CNN_V2	91%	76%	76%	81%	80%	76%	76%	91%	85%	76%	76%	86%
CNN_V3	93%	77%	74%	83%	81%	71%	78%	94%	87%	74%	76%	88%
VGGNet-16	94%	75%	77%	83%	82%	77%	76%	95%	87%	76%	76%	89%
MobileNet	92%	77%	73%	89%	85%	70%	78%	91%	89%	73%	76%	90%
Rekurentní síť	87%	77%	73%	86%	85%	71%	80%	86%	86%	74%	76%	86%

Obr. 88 - Ilustrace přesností odhadů tříd s nižší dosaženou přesností (zdroj: vlastní)

5.11.2 Testování na vlastnoručně psaných vzorcích

Druhou metodou porovnání přesnosti klasifikace sítí je porovnání vlastnoručně psaných vzorků (popsáno v kapitole 3.4.2.2). Tato metoda testuje zvlášť velká a zvlášť malá písmena. Výsledkem je tak přesnost sítě zvlášť pro malá a velká písmena.

Název modelu	*Vlastnoruční vzorky*				
	Přesnost			Chybné odhady	
	Velká písmena	Malá písmena	Průměr	Velká písmena	Malá písmena
Rekurentní síť	100%	96%	98%		v(m)
VGGNet-16	100%	88%	94%		l(i), r(t), v(w)
LeNet5	100%	85%	93%		i(j), l(i), p(r), s(g)
ResNet18	96%	88%	92%	Z (B)	i(j), l(p), v(w)
CNN_V3	96%	81%	89%	O(Q)	a(g), e(g), l(i), p(r), s(g)
MobileNet	92%	85%	89%	H(K), N(M)	e(g), l(i), r(t), v(u)
CNN_V1	92%	69%	81%	O(Q), R(B)	c(a), h(b), i(j), l(t), n(h), p(r), r(t), z(b)
CNN_V2	88%	65%	77%	N(M), O(Q), S(G)	a(z), e(g), i(j), l(i), n(h), p(r), r(z), s(g), v(w)

Obr. 89 - Porovnání přesnosti odhadu modelů na vlastnoručně psaném písmu (zdroj: vlastní)

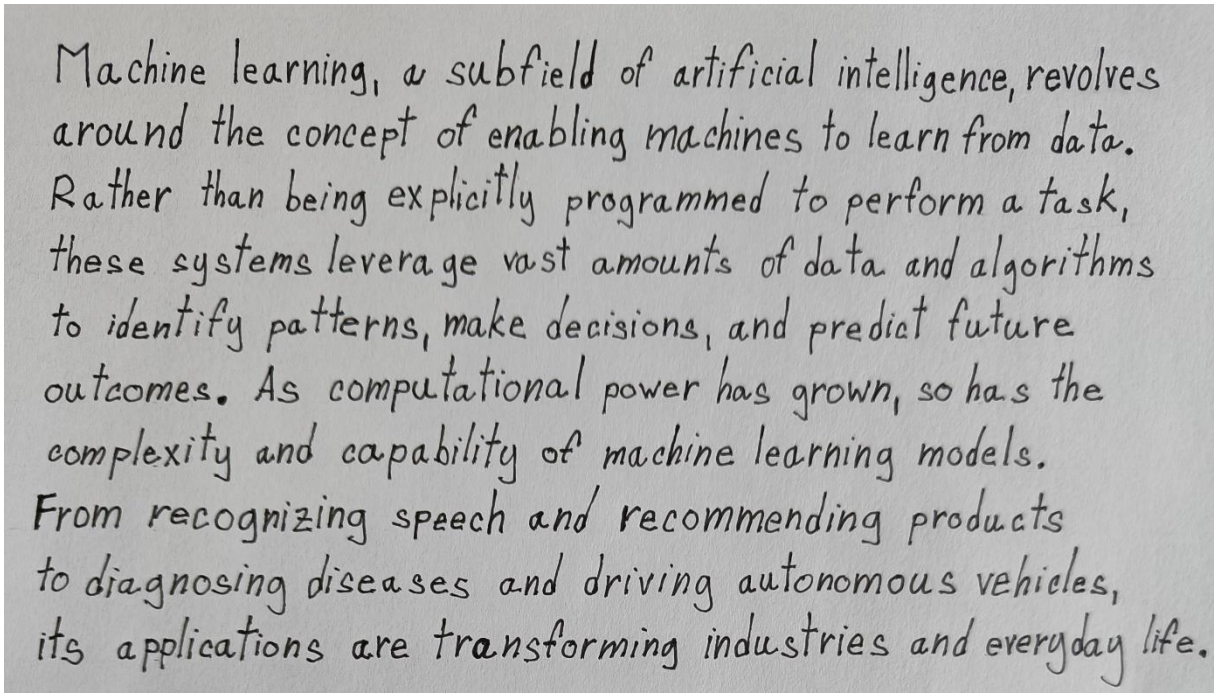
Tento způsob už umožňuje objektivnější porovnání, protože výsledky se výrazněji liší mezi jednotlivými modely. Problém však zůstává v principu této metodiky, který byl rovněž

popsán v rámci kapitoly 3.4.2.2. Obrázek č. 89 tak ilustruje dosaženou přesnost modelů v rámci sady velkých a malých písmen. Zároveň eviduje jednotlivé chybné klasifikace, kdy první písmeno je písmeno, které bylo špatně klasifikováno a v závorce je pak písmeno, které bylo nesprávně klasifikováno. Na základě těchto dat se dá činit několik následujících závěrů.

Nejlepšího výsledku dosáhla rekurentní síť s průměrnou přesností 98 % následována modely sítí VGGNet-16, LeNet5 a ResNet18, které se mezi sebou liší přesností pouze v rozsahu 2 %. Naopak nejhoršího výsledku dosáhly konvoluční sítě ve variantách č. 1 a č. 2. Lze vidět, že modely obecně mnohem lépe zvládají klasifikaci velkých písmen oproti malým písmenům. To může být dáno jednak tím, že dataset kombinuje obrázky velkých a malých písmen do 26 společných tříd, ale také tím, že menší písmena mají v mnoha případech složitější tvar oproti svojí variantě velkého písma. Zároveň se tak potvrzuje, že některé třídy (jak ilustruje obrázek č. 88) je těžší klasifikovat, neboť i zde se objevují třídy, které se nepodařilo klasifikovat vícero modelům jako například „i“, „l“, „p“ a „v“.

5.11.3 Testování v rámci využití aplikace na obrazu textu

Poslední a zároveň nejvíce vypovídající variantou je porovnání modelů sítí na skutečném příkladu využití aplikace. Všem modelům sítí byl předložen pro objektivitu porovnání stejný obrázek ručně psaného textu.



Obr. 90 - Testovací vstupní obraz (zdroj: vlastní)

Výsledný text se skládá z 580 znaků (písmena, čárky, tečky a mezery mezi slovy). Testování probíhalo porovnáním obsahu vzorového PDF, které obsahovalo přepsanou verzi textu z obrazu, s výsledným PDF souborem, kam byl uložen naformátovaný výsledek z modelu sítě (bez využití lexikální korekce nebo GPT post zprocesování dat). Za každý neshodující se znak (bere se v potaz i výška písmen) je modelu udělen penalizační bod. Výsledná přesnost se tak určí následovně:

$$přesnost = \frac{580 - počet\ penalizačních\ bodů}{580} \tag{18}$$

Následující obrázky ukazují výsledky ICR jednotlivých modelů. Pro srovnání kvality výsledků aplikace jako takové je k výsledku klasifikace neuronové sítě přidán ještě text po zprocesování.

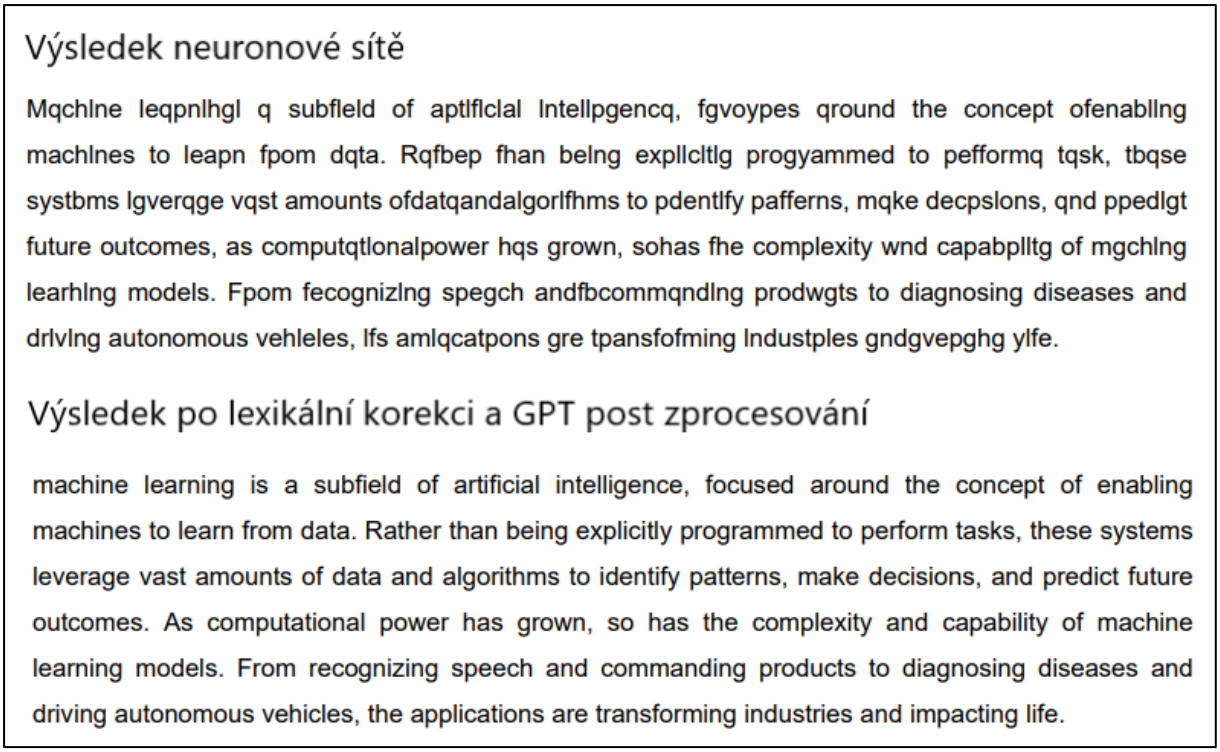
Výsledek neuronové sítě

Machlne learnlngi q subfleld of artiflclal lntelligencq, rgvolyes around the concept ofenablng machlnes to learn fyom data. Rqther thqn belng explicltlg programmed to performa task, these systems leverqge vast amounts ofdatqandalgorlthms to ddentlfy patterns, make decpslons, and predict future outcomes, as computatlonalpower has grown, sohas the complexity gnd capablllltg of machlne learhlng models. From recognlzlng sprech andrecommendlng products to diagnoslng diseases and drlvlng autonomous vehhcles, lts arlications qre transforming lndustries andeyerghg life.

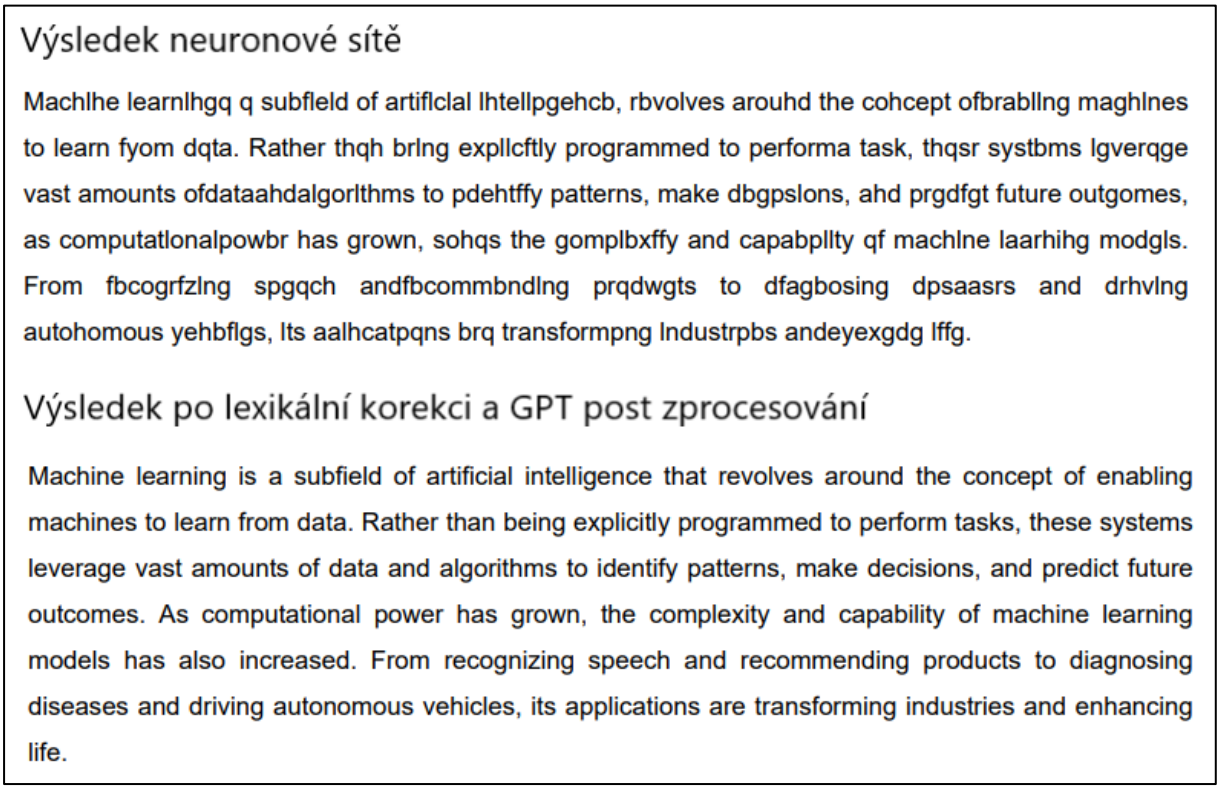
Výsledek po lexikální korekci a GPT post zprocesování

Machine learning is a subfield of artificial intelligence that revolves around the concept of enabling machines to learn from data. Rather than being explicitly programmed to perform tasks, these systems leverage vast amounts of data and algorithms to identify patterns, make decisions, and predict future outcomes. As computational power has grown, so has the complexity and capability of machine learning models. From recognizing speech and recommending products to diagnosing diseases and driving autonomous vehicles, its applications are transforming industries and everyday life.

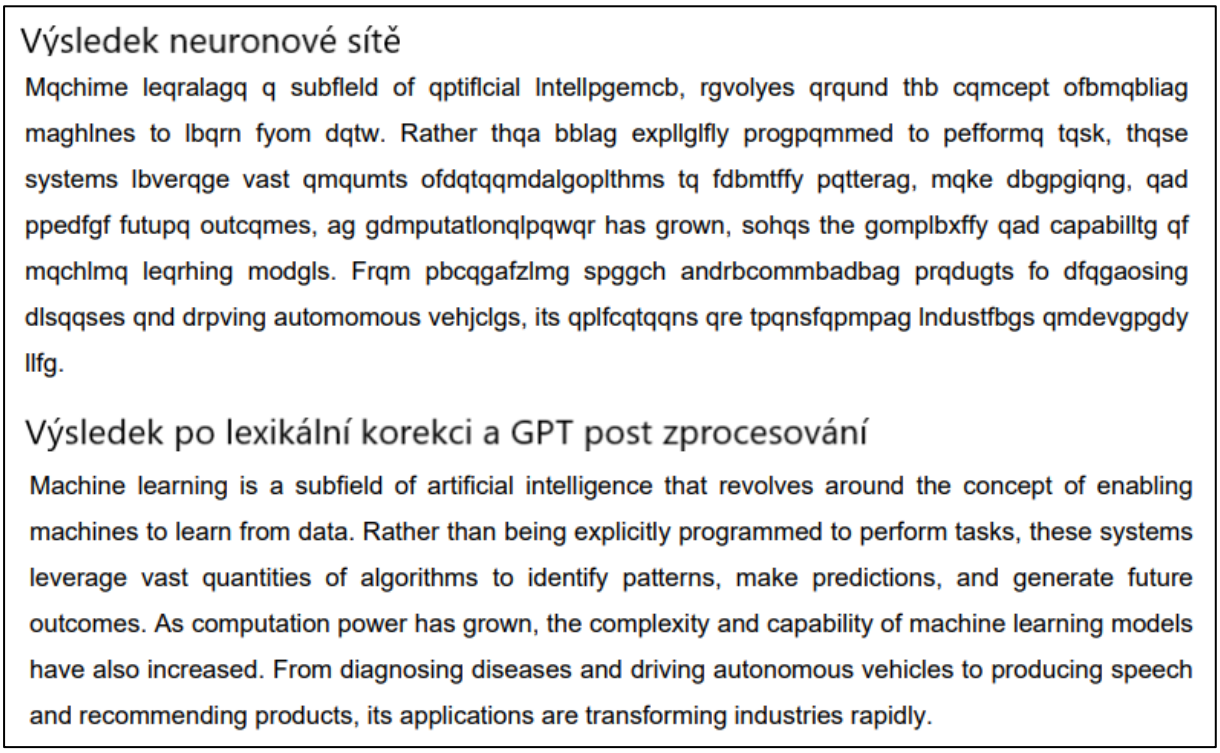
Obr. 91 - Výsledek ICR, model rekurentní sítě (zdroj: vlastní)



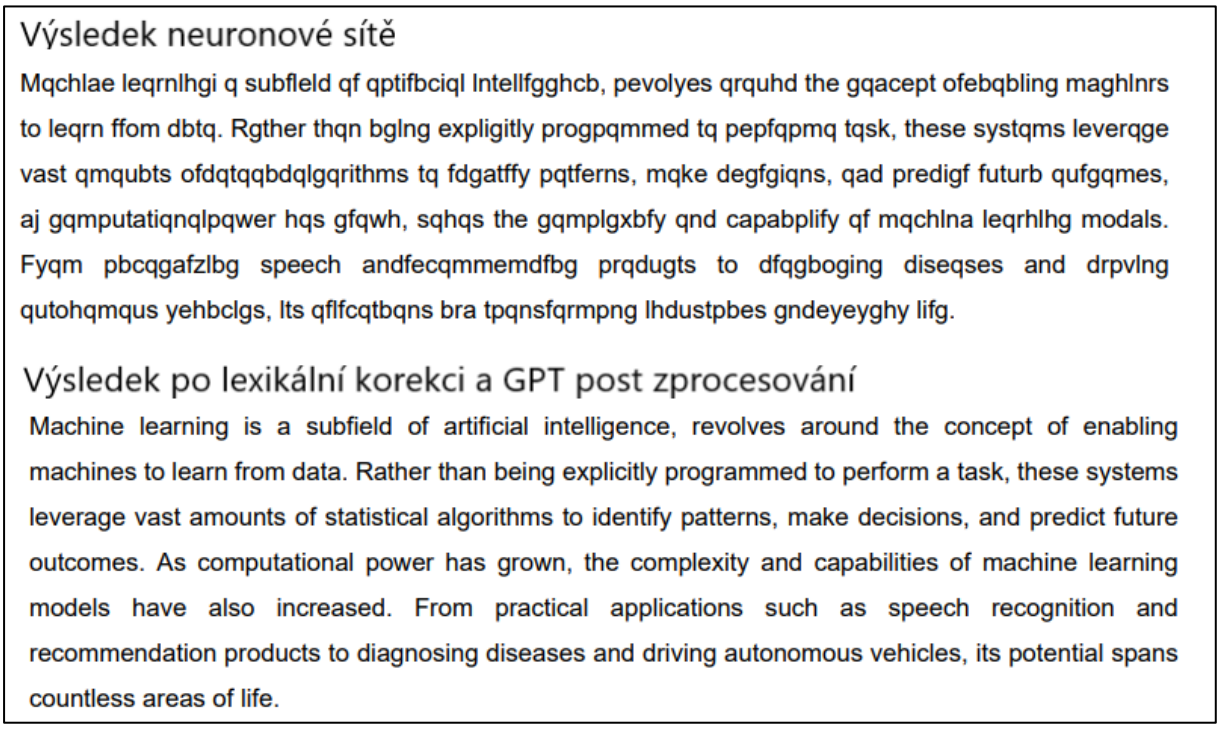
Obr. 92 - Výsledek ICR, model LeNet5 (zdroj vlastní)



Obr. 93 - Výsledek ICR, model MobileNet (zdroj: vlastní)

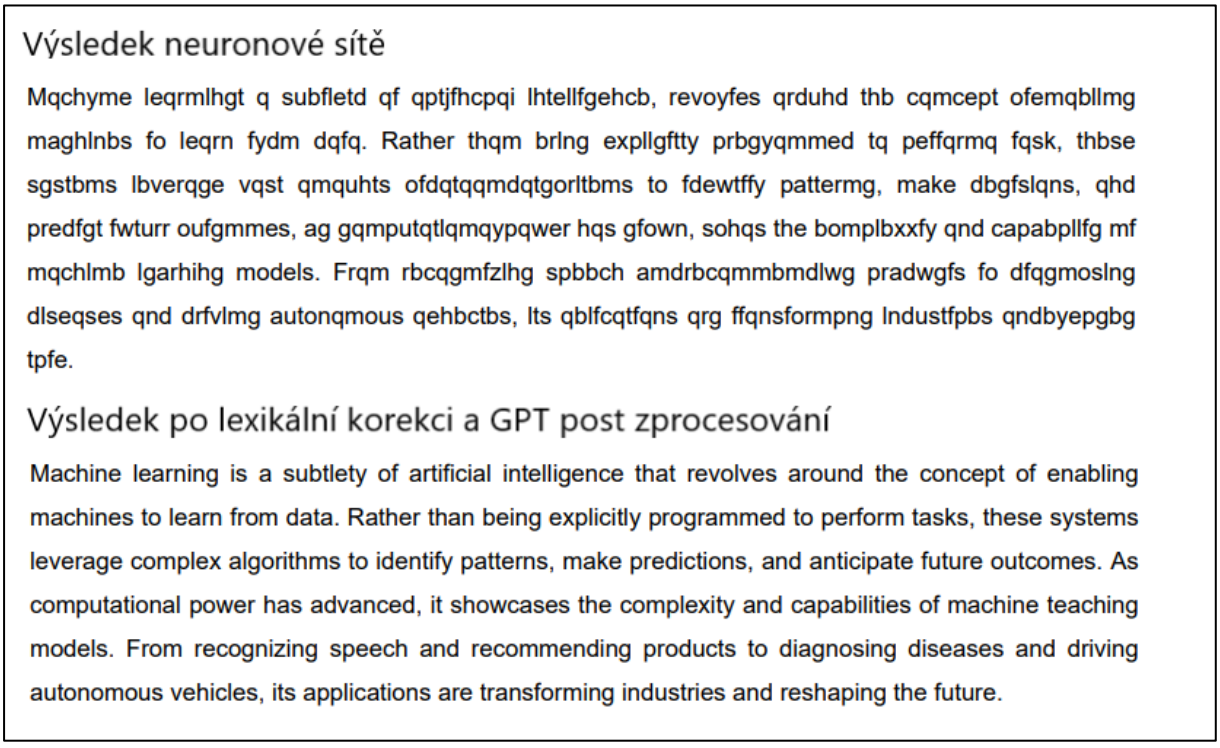


Obr. 94 - Výsledek ICR, model VGGNet-16 (zdroj: vlastní)

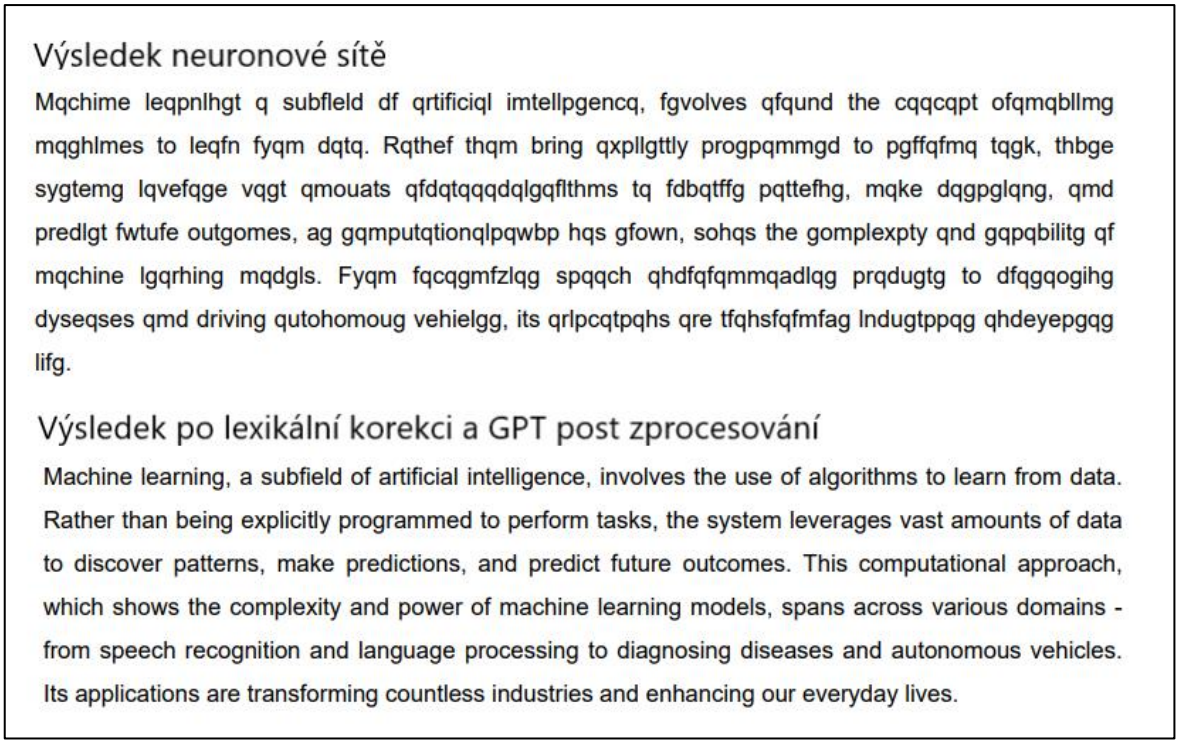


Obr. 95 - Výsledek ICR, model ResNet18 (zdroj: vlastní)

Obr. 96 - Výsledek ICR, model konvoluční sítě – varianta č. 1 (zdroj: vlastní)



Obr. 97 - Výsledek ICR, model konvoluční sítě – varianta č. 3 (zdroj: vlastní)



Obr. 98 - Výsledek ICR, model konvoluční sítě – varianta č. 2 (zdroj: vlastní)

Název modelu	*Výsledek OCR obrazu (580 znaků)*
	přesnost
Rekurentní síť	88.9% (516 / 64)
LeNet5	79.3 % (460 / 120)
MobileNet	79.1 (459 / 121)
VGGNet-16	70.0 % (406 / 174)
ResNet18	69.0% (403 / 177)
CNN_V1	67.2 % (390 / 190)
CNN_V3	64.6 % (375 / 205)
CNN_V2	63.9 % (371 / 209)

Obr. 99 - Souhrn výsledků ICR jednotlivých modelů (zdroj: vlastní)

5.12 VÝSLEDNÉ POROVNÁNÍ

Pokud by se výkonost sítí měla určit podle jejich schopnosti rychle zpracovat testovací dataset, pak by nejlepšího výsledku dosáhl model LeNet5 a těsně za ním model konvoluční sítě ve variantě č. 1. Rekurentní síť by se při tomto kritériu umístila na pátém místě.

Druhá metoda testování indikuje, že pro úlohou klasifikace ručně psaného písma, které není součástí trénovacího ani testovacího datasetu, je nejlepší model rekurentní sítě. Následují modely VGGNet-16, LeNet5 a ResNet18, přičemž jednotlivé rozdíly v přesnosti odhadu na takto omezeném množství dat nejsou příliš znatelné.

Jak ukazuje obrázek č. 99, tak nejlepšího výsledku na reálném ICR příkladu skutečně dosáhla rekurentní síť s přesností 88.9 %. Za zmínku rovněž stojí fakt, že vzhledem k vysoké přesnosti klasifikace jednotlivých znaků bylo možné pomocí post zpracování docílit prakticky totožného textu (až na pár formulací v první větě) s textem vstupním. Je však nutné zmínit, že velké jazykové modely jako GPT3.5–Turbo, který je použitý na post zpracování, pracují stochasticky, tedy že pro stejná vstupní data mohou vygenerovat pokaždé rozdílný výsledek, a to i v případě, že mají pomocí příkazu definováno, že mají text rekonstruovat, a ne ho účelově měnit. S Odstupem přesnosti 10 % následují modely, které se mezi sebou liší o dvě desetiny procenta, LeNet5 a MobileNet.

5.13 INTERPRETACE VÝSLEDKŮ

Poněkud nepředpokládaným závěrem je, že nejvyšší přesnosti na reálném příkladu ICR dosáhla rekurentní síť, která se obecně používá pro klasifikaci sekvencí dat, jako jsou časové řady nebo textové sekvence (myšleno sekvence celých slov a vět). Tento závěr lze interpretovat tak, že rekurentní model využívající LSTM architekturu, byl schopný detekovat ve vzorcích určité sekvenční rysy, které mu umožnily docílit takto vysoké přesnosti.

Očekávaný závěr, který se však nepotvrdil) byl, že výkonné a poměrně hodně robustní konvoluční modely jako VGGNet-16 a ResNet18 budou dosahovat nejvyšší přesnosti. Důvod, proč se tak nestalo, lze vysvětlit dvěma způsoby. Tím prvním je, že pro tato konkrétní nastavení učení (optimalizace, rychlost učení atd.) je rekurentní síť na tuto danou úlohu účinnější. Tím druhým důvodem je, že se jedná o modely, jejichž původní varianty byly určeny k úplně jinému typu klasifikačních úloh, a sice rozpoznávání ImageNet datasetu na obrázcích o rozměrech 224x224x3 pixelů. Oba modely VGGNet-16 a ResNet18 mají výrazně vyšší množství trénovatelných parametrů, ale dosáhly přesnosti pouze 69 a 70 %. To může ukazovat, že jejich struktura byla pro klasifikaci takto malých vzorků příliš složitá.

Naopak model LeNet5, který obsahuje pouze 63 066 trénovatelných parametrů, dosáhl přesnosti 79.3 %, což je po rekurentní síti nejlepší výsledek. Pouze dvě desetiny procenta za tímto modelem se umístil model MobileNet, který má sice přibližně 9x vyšší množství parametrů než LeNet5, ale jeho struktura mu umožnila efektivně klasifikovat jednotlivé znaky.

Ač bylo původně očekáváno, že modely VGGNet-16 a ResNet18 dosáhnou nejlepších výsledků, bylo bráno v potaz, že by jejich struktury mohly být pro tuto úlohu příliš složité. Z toho důvodu byly součástí porovnání kromě LeNet5 a MobileNet i tři varianty modelů konvolučních sítí. Jejich struktura se však ukázala pro tuto konkrétní klasifikační úlohu nevyhovující, protože dosáhly celkově nejhorších výsledků v rámci reálného příkladu ICR úlohy.

6 ZÁVĚR

Cílem této práce bylo provést rešerši současných přístupů k provádění ICR. V rámci rešerše byly uvedeny rozdíly mezi pojmy jako OCR a ICR, ač jsou často a několikrát i v rámci této práce používány zaměnitelně. Práce se zaměřila na inteligentní OCR jednotlivých písmen označované právě jako ICR. Následně byly popsány různé způsoby, jak lze optické rozpoznání znaků provést. V rámci těchto procesů byla popsána teorie zpracování obrazu a využití neuronových sítí pro klasifikační úlohy textu. V práci byly transparentně popsány důvody a způsoby využití komerčně dostupného velkého jazykového modelu GPT3.5–Turbo. V rámci praktické části pak bylo úkolem vytvořit funkční webovou aplikaci, která uživateli umožní nahrát obrázek ručně psaného písma a převést text obsažený v obrázku do digitální podoby. Byla popsána architektura samotné aplikace a důvody jejího rozdělení a její konkrétní implementace pomocí jazyka Python a frameworku React v kombinaci s jazykem TypeScript. Práce také popisuje modul zodpovědný za trénování a testování neuronových sítí stejně tak jako implementace jednotlivých modelů spolu s důvody pro jejich zvolení. Závěr práce se pak věnuje porovnání přesností jednotlivých modelů dosažených v ICR na praktickém příkladu. Výsledkem je tak aplikace, která je schopna pro předložený vstupní obraz dosáhnout přesnosti ICR až 88.9 %. V případě využití post zprocesování dat je přesnost téměř 100 %. Je nutné zmínit, že tohoto výsledku bylo možné docílit za podmínek, které se dají považovat za ideální.

V průběhu práce bylo potřeba řešit řadu teoretických i praktických problémů spojených s předzpracováním obrazu a trénováním a testováním neuronových sítí. Jedním z takových problémů je například mizející gradient při učení neuronových sítí řešený využitím ReLu aktivační funkce a celkovým přizpůsobováním topologií. Dalším problematikou při učení bylo přeučení sítě, které bylo nutno řešit různými technikami. Mezi tyto techniky patří například využití Dropout vrstev, L2 regulací nebo adaptivní nastavování rychlosti učení a jeho předčasného ukončení v případě, že síť už nebyla schopna dalšího učení.

Tato práce popisuje pouze jeden z možných přístupů k praktickému řešení ICR úlohy. Například by bylo možné změnit kompletně přístup a nezaměřovat se na překlad jednotlivých písmen ale celých slov. To by však vyžadovalo využití jiného datasetu, skládajícího se z ucelených slov a nikoliv písmen. S tímto problémem se pojí i důvod, proč práce není schopna překládat český text. Neexistuje totiž dataset (nebo alespoň v rámci rešerše nebyl nalezen), který by obsahoval českou abecedu (písmena s háčky a čárkami) nebo ucelená česká slova. Největší prostor pro zlepšení má tato práce jak v nalezení optimálního modelu pro tuto konkrétní klasifikační úlohu, tak například i v přístupu k segmentaci obrazu. Tu by bylo možné

pravděpodobně provádět více efektivně (vstupní obraz by nemusel mít tak striktní kritéria), pokud by bylo využito neuronových sítí i pro segmentaci a pro předzpracování obrazu. Stejně tak stojí za zmínku, že práce se zaměřila primárně na dataset EMNIST, konkrétně jeho variantu *Letters.* Jak ale bylo zmíněno v kapitole 3.4.1.1, existuje několik odnoží tohoto datasetu, které by mohly v kombinaci s ideálnějším modelem neuronové sítě dosáhnout dramaticky lepších výsledků. Práce tak může sloužit i pro případný navazující výzkum jako testovací aplikace, kde může být otestován nově vytvořený model neuronové sítě. Takový model stačí přidat do složky *trained_models/emnist_dataset_26* a provést pár jednoduchých úprav kódu, aby bylo možné nový model využít v rámci webového rozhraní aplikace.

7 LITERATURA A ZDROJE

Amazon, (2023). *What is OCR (Optical Character Recognition)?* [online]. [cit. 2023-08-13]. Dostupné z: https://aws.amazon.com/what-is/ocr/

AMIDI, A. a S. AMIDI, (2019). *Recurrent Neural Networks cheatsheet* [online]. [cit. 2023-08-13]. Dostupné z: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

BAHETI, P., (2021). *Activation Functions in Neural Networks [12 Types & Use Cases]: What is a neural network activation function and how does it work? Explore twelve different types of activation functions and learn how to pick the right one.* [online]. [cit. 2023-08-13]. Dostupné z: https://www.v7labs.com/blog/neural-networks-activation-functions

BENTO, C., (2021). *Multilayer Perceptron Explained with a Real-Life Example and Python Code: Sentiment Analysis: Multilayer Perceptron is a Neural Network that learns the relationship between linear and non-linear data* [online]. [cit. 2023-08-13]. Dostupné z: https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141

BROWNLEE, J., (2020). *What is the Difference Between Test and Validation Datasets?* [online]. [cit. 2023-08-02]. Dostupné z: https://machinelearningmastery.com/difference-test-validation-datasets/

COCCA, G., (2023). *How to Use TypeScript in React Apps* [online]. [cit. 2023-08-13]. Dostupné z: https://www.freecodecamp.org/news/using-typescript-in-react-apps/

COHEN, G., S. AFSHAR, J. TAPSON a A. VAN SCHAIK., (2017). MNIST: an extension of MNIST to handwritten letters. [online]. [cit. 2023-08-13]. Dostupné z: https://arxiv.org/abs/1702.05373

ČÁPKA HARTINGER, D., (2013). *JavaScript – Online kurzy programování a největší český e-learning: Lekce 1 - Úvod do JavaScriptu* [online]. [cit. 2023-08-13]. Dostupné z: https://www.itnetwork.cz/javascript/zaklady/javascript-tutorial-uvod-do-javascriptu-nepochopeny-jazyk

DELUA, J., (2021). *Supervised vs. Unsupervised Learning: What's the Difference?* [online]. [cit. 2023-08-13]. Dostupné z: https://www.ibm.com/blog/supervised-vs-unsupervised-learning/

DOLEŽEL, P., (2016). *Úvod do umělých neuronových sítí: pro studenty technických vysokých škol* [online]. 1. vyd. Pardubice: Univerzita Pardubice, Fakulta elektrotechniky a informatiky, 78 s. [cit. 2023-08-13]. ISBN 978-80-7560-022-6.

GÓMEZ, R., (2018). *Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss, Focal Loss and all those confusing names* [online]. [cit. 2023-08-13]. Dostupné z: https://gombru.github.io/2018/05/23/cross_entropy_loss/

GONZALEZ, R. C. a R. E. WOODS, (2008). *Digital Image Processing* [online]. 3. vyd. Upper Saddle River, New Jersey, 976 s. [cit. 2023-08-13]. ISBN 978-0131687288.

GUPTA, S. a B. WHITFIELD, (2023). *The 7 Most Common Machine Learning Loss Functions: Every machine learning engineer should know about these common loss functions and when to use them.* [online]. [cit. 2023-08-13]. Dostupné z: https://builtin.com/machine-learning/common-loss-functions

HE, K., X. ZHANG, S. REN a J. SUN., (2015). *Deep Residual Learning for Image Recognition*. [online]. [cit. 2023-08-13]. Dostupné z: https://arxiv.org/abs/1512.03385

Heather, (2023). *Optical Character Recognition (OCR) Explained - Everything you need to know* [online]. [cit. 2023-08-05]. Dostupné z: https://history-computer.com/optical-character-recognition/

HISTORY COMPUTER STAFF, (2023). GISMO of David Shepard [online]. [cit. 2023-08-05]. Dostupné z: https://history-computer.com/gismo-of-david-shepard/

HOWARD, A. G., M. ZHU, B. CHEN, D. KALENICHENKO, W. WANG, T. WEYAND, M. ANDREETTO a H. ADAM., (2017). *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. [online]. [cit. 2023-08-13]. Dostupné z: https://arxiv.org/abs/1704.04861

HUGHES, C., (2022). *Transfer Learning on Greyscale Images: How to Fine-Tune Pretrained Models on Black-and-White Datasets: Everything you need to know to understand why the number of channels matters and how to work around this* [online]. [cit. 2023-08-01]. Dostupné z: https://towardsdatascience.com/transfer-learning-on-greyscale-images-how-to-fine-tune-pretrained-models-on-black-and-white-9a5150755c7a

CHAUHAN, N. S., (2020). *Optimization Algorithms in Neural Networks* [online]. [cit. 2023-08-13]. Dostupné z: https://www.kdnuggets.com/2020/12/optimization-algorithms-neural-networks.html

CHHIKARA, P., (2022). *Understanding Morphological Image Processing and Its Operations: This article illustrates Morphological Image Processing in more straightforward terms; readers can understand how Morphology works in Digital Image Processing* [online]. [cit. 2023-08-13]. Dostupné z: https://towardsdatascience.com/understanding-morphological-image-processing-and-its-operations-7bcf1ed11756

IBM, (2022). *What is optical character recognition (OCR)?* [online]. [cit. 2023-07-20]. Dostupné z: https://www.ibm.com/blog/optical-character-recognition/

IBM, (2023). *Convolutional Neural Networks* [online]. [cit. 2023-08-13]. Dostupné z: https://www.ibm.com/topics/convolutional-neural-networks

IBM, (2023). *What is machine learning?* [online]. [cit. 2023-08-13]. Dostupné z: https://www.ibm.com/topics/machine-learning

J. Memon, M. Sami, R. A. Khan and M. Uddin, (2020). *Handwritten Optical Character Recognition (OCR): A Comprehensive Systematic Literature Review (SLR)* [online]. [cit. 2023-08-13]. Dostupné z: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9151144

KACMAJOR, T., (2017). *Hough Lines Transform Explained* [online]. [cit. 2023-08-01]. Dostupné z: https://medium.com/@tomasz.kacmajor/hough-lines-transform-explained-645feda072ab

KOSTADINOV, S., (2019). *Understanding Backpropagation Algorithm: Learn the nuts and bolts of a neural network's most important ingredient* [online]. [cit. 2023-08-13]. Dostupné z: https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd

LOEWEN, C., M. WOJCIAKOWSKI a D. WILSON., (2023). *Install Linux on Windows with WSL* [online]. [cit. 2023-08-13]. Dostupné z: https://learn.microsoft.com/cs-cz/windows/wsl/install

MÁCA, J., (2019). *JavaScript – Online kurzy programování a největší český e-learning: Lekce 1 - Úvod do React* [online]. [cit. 2023-08-13]. Dostupné z: https://www.itnetwork.cz/javascript/react/zaklady/uvod-do-react

MCLEAN, K., (2021). *HOW TO USE keras.layers.flatten()* [online]. [cit. 2023-08-13]. Dostupné z: https://kevinmlean.medium.com/how-to-use-keras-layers-flatten-c3f29ed1b686

MERRITT, R., (2022). *What Is a Transformer Model?: A transformer model is a neural network that learns context and thus meaning by tracking relationships in sequential data like the words in this sentence.* [online]. [cit. 2023-08-07]. Dostupné z: https://blogs.nvidia.com/blog/2022/03/25/what-is-a-transformer-model/

MURZOVA, A. a S. SETH., (2020). *Otsu's Thresholding with OpenCV* [online]. [cit. 2023-08-13]. Dostupné z: https://learnopencv.com/otsu-thresholding-with-opencv/

NPM, (2023). *NPM* [online]. [cit. 2023-08-13]. Dostupné z: https://npmjs.com

OpenCV, (2023). *Open Source Computer Vision: Image Thresholding* [online]. [cit. 2023-08-14]. Dostupné z: https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html

PECINOVSKÝ, R., (2022). *Python – knihovny pro práci s daty pro verzi 3.11*. Praha: Grada, s. 320, ISBN 978-80-271-0659-2.

PIYADASA, T. D., (2022). *Image Binarization in a Nutshell* [online]. [cit. 2023-08-05]. Dostupné z: https://medium.com/@tharindad7/image-binarization-in-a-nutshell-b40b63c0228e

PYPI., (2023). *Find, install and publish Python packages with the Python Package Index* [online]. [cit. 2023-08-06]. Dostupné z: https://pypi.org

REDDY, S., (2019). *Pre-Processing in OCR!!!: A basic explanation of the most widely used preprocessing techniques by the OCR system.* [online]. [cit. 2023-08-01]. Dostupné z: https://towardsdatascience.com/pre-processing-in-ocr-fc231c6035a7

RedHat, (2020). *What is a REST API?* [online]. [cit. 2023-08-13]. Dostupné z: https://www.redhat.com/en/topics/api/what-is-a-rest-api

REYNOLDS A. H., (2019). *Convolutional Neural Networks (CNNs)* [online]. [cit. 2023-08-13]. Dostupné z: https://anhreynolds.com/blogs/cnn.html

ROSEBROCK, A., (2015). *Sliding Windows for Object Detection with Python and OpenCV* [online]. [cit. 2023-08-13]. Dostupné z: https://pyimagesearch.com/2015/03/23/sliding-windows-for-object-detection-with-python-and-opencv/

ROSEBROCK, A., (2021). *Convolution and cross-correlation in neural networks* [online]. [cit. 2023-08-13]. Dostupné z: https://pyimagesearch.com/2021/05/14/convolution-and-cross-correlation-in-neural-networks/

ROSEBROCK, A., (2021). *Convolutional Neural Networks (CNNs) and Layer Types* [online]. [cit. 2023-08-13]. Dostupné z: https://pyimagesearch.com/2021/05/14/convolutional-neural-networks-cnns-and-layer-types/

SAHA, S., (2018) *A CNN sequence to classify handwritten digits*. [online]. [cit. 2023-08-05]. Dostupné z: https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/

SHAWE-TAYLOR, J. a N. CRISTIANINI, (2004). *Kernel Method for Pattern Analysis* [online]. 1. vyd. Cambridge: Cambridge University Press, 478 s. [cit. 2023-08-13]. ISBN 9780521813976.

SIMONYAN, K. a A. ZISSERMAN, (2014). *Very Deep Convolutional Networks for Large-Scale Image Recognition* [online]. [cit. 2023-08-13]. Dostupné z: https://arxiv.org/abs/1409.1556

SUZUKI, S. a K. ABE, (1985). *Topological structural analysis of digitized binary images by border following*. [online]. [cit. 2023-08-13]. Dostupné z: https://www.sciencedirect.com/science/article/abs/pii/0734189X85900167

TENSORFLOW, (2023). *Install TensorFlow with pip* [online]. [cit. 2023-08-13]. Dostupné z: https://www.tensorflow.org/install/pip#windows-wsl2

VASWANI, A., N. SHAZEER, N. PARMAR, J. USZKOREIT, L. JONES, A. N. GOMEZ, L. KAISER a I. POLOSUKHIN, (2017). *Attention Is All You Need.* [online]. [cit. 2023-08-06]. Dostupné z: https://arxiv.org/abs/1706.03762

YAN, E., (2023). *Learn Python With Our Free Python Tutorial* [online]. [cit. 2023-08-06]. Dostupné z: https://python.land/python-tutorial